

# Домашна задача 2

Тијана Атанасовска (196014)

## Задача 1

### А)

Да се напише предикат во PROLOG `rodeni_razlicen_grad(Kolku)` кој враќа колку лица се родени во град различен од градовите на раѓање на двата родитела.

За таа цел потребно е за секое лице за кое има информација за неговите родители во базата на податоци да се пристапи до градовите на раѓање и да се споредат. Потоа да се изброи за колку лица е исполнет предикатот.

Се користат следните предикати:

- `dali_e_razlicen(GradDete, GradRoditel1, GradRoditel2)` – За градовите на раѓање на родителите и на детето, се враќа `True/False` зависно дали се различни.

```
32 dali_e_razlicen(GradDete, GradRoditel1, GradRoditel2):-  
33     GradDete\=GradRoditel1,GradDete\=GradRoditel2.  
34
```

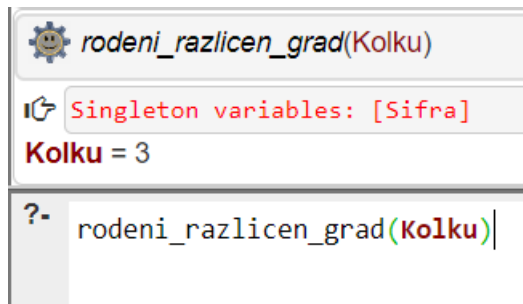
- `razlicni(_)` – За секој запис од тип `familija`, се пристапува до шифрите на Родител1, Родител2 и листата од шифрите на децата. Се изминуваат шифрите на децата со функцијата `member`. Во `Dete` се чува само моменталната шифра на едно дете до кое е стигнато изминувањето на листата. Се пристапува до градовите на раѓање на детето и неговите родители преку предикатот `lice`. Шифрите ги задаваме како прв аргумент во `lice`, бидејќи таква е структурата во базата на податоци. За останатите информации од структурата кои не се потребни се става `_`. Откако ќе ги земеме градовите во променливите `GradDete`, `GradRoditel1`, `GradRoditel2`, се проверуваат дали се различни со повикување на горниот предикат `dali_e_razlicen`.

```
38 razlicni(_):-familija(R1,R2,Deca), member(Dete,Deca), lice(Dete,_,_,_,GradDete,_),  
39     lice(R1,_,_,_,GradRoditel1,_), lice(R2,_,_,_,GradRoditel2,_),  
40     dali_e_razlicen(GradDete,GradRoditel1,GradRoditel2).
```

- `rodeni_razlicen_grad(Kolku)` – Главниот предикат кој со `findall` го повикува предикатот `razlicni(_)`. За оние вредности во `familija` за кои предикатот е исполнет зачувува мемориски локации во `List` и потоа во излезната променлива `Kolku` ја враќа должината на листата.

```
45 rodeni_razlicen_grad(Kolku):-findall(_,razlicni(_),List),length(List,Kolku).|
```

Повикување на предикатот:



```
rodeni_razlicen_grad(Kolku)
Singleton variables: [Sifra]
Kolku = 3
?- rodeni_razlicen_grad(kolku)|
```

## Б)

Да се напише предикат во PROLOG `predci(Sifra,L)` кој на влез прима шифра на некое лице и за тоа лице на излез треба да ги врати сите негови предци (нивните шифри) за кои се исполнети следните услови: предокот да биде од истиот пол како и лицето и денот на раѓање на предокот да биде најмногу една недела пред или после денот на раѓање на лицето (годината е небитна и претпоставете дека сите месеци имаат по 30 денови).

Потребно е да се најдат сите предци на едно лице од ист пол, а потоа да се споредат нивните родендени и полот. За некој да биде предок мора да е директен Родител или родител на родител, итн рекурзивно.

- `roditeli(Sifra,R1,R2)` – Предикат за наоѓање на директни родители со пристап до структурата на фамилија. Не може директно да се провери дали некој е родител на дете со дадена шифра, бидејќи децата се дадени во листа, па мора да се итерира. Тоа се постигнува со `member`.

```
51 roditeli(Sifra,R1,R2):-familija(R1,R2,Deca),member(Sifra,Deca).
```

- `najdi_predci(Sifra,Predci)` – Рекурзивен предикат за наоѓање на предци. Првин се наоѓаат директните родители на лицето со дадена Шифра (влезен арг). Тие се лепат на излезната како први два членови, а другиот дел од листата `Result`, ќе се генерира подоцна во предикатот. Потоа за секој од родителите се повикува рекурзивно да се најдат нивните предци. Резултатите запишани како листи во `Subresult1`, `Subresult2` се спојуваат во една листа и се запишуваат во `Result`. Мора да се повика `append` бидејќи подрезултатите се листи, па не може директно да се залепат после `R1,R2`.

`Cut` операторот мора да се постави на позицијата поради природата на Пролог, која се враќа назад за да провери дали за други променливи важат предикатите. Ова резултира со појава на дупликации.

Основниот повик е после рекурзивниот повик. Доколку е пред него би се враќале само директните родители. (Објаснување: Рекурзијата почнува и се наоѓаат директните родители со предикатот `roditeli`. За секој родител се повикува

najdi\_predci, но ако основниот повик е пред рекурзивниот овие повици ќе вратат празна листа, бидејќи Пролог унифицира променлива со било што, па така и со празна листа.)

```
60 najdi_predci(Sifra,[R1,R2|Result]):-roditeli(Sifra,R1,R2),
61     najdi_predci(R1,Subresult1), najdi_predci(R2,Subresult2),!,
62     append(Subresult1,Subresult2,Result).
63 najdi_predci(Sifra,[]).
```

- **rodendeni(Sifra, SifraPredok)** – Проверува дали за дадена Шифра на лица и Шифра на негов предок, соодветните родендени се со разлика до 7 дена. За таа цел првин се пристапува до денот и месецот на роденденот (податоци кои се наоѓаат во структурата lice, па мора да се земаат преку неа). За да бидат родендените во разлика од 7 дена, тогаш месеците мора да бидат исти или да се соседни месеци (разлика 1). Таа проверка се прави со линија 67. Доколку ова е задоволено, следно е да се провери разликата од деновите. Ако месецот е ист, деновите се апсолутна разлика. Ако месецот е различен, потребно е да знаеме кој месец е помалиот за тој ден да го одземеме од 30, а деновите од поголемиот месец само да се додадат. Пример: 26.11 4.12 -> (30-26)+4

Мора да се напишани и двете линии 69 и 70 бидејќи не знаеме кај кое од лицата е поголемиот месец.

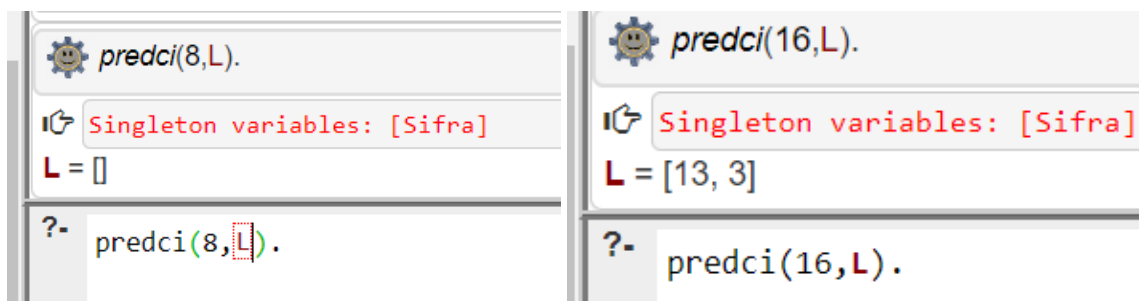
Овие три услови се разделени со оператор ИЛИ (;) бидејќи може да е исполнет само еден од нив (а тоа е и доволно).

```
65 rodendeni(Sifra, SifraPredok):-lice(Sifra,_,_,_,datum(Den1,Mesec1,_),_,_),
66     lice(SifraPredok,_,_,_,datum(Den2,Mesec2,_),_,_),
67     Meseci is abs(Mesec1-Mesec2),(Meseci is 0;Meseci is 1),
68     (    (Mesec1==Mesec2,Denovi is abs(Den1-Den2));
69         (Mesec1>Mesec2, Denovi is ((30-Den2)+Den1));
70         (Mesec1<Mesec2, Denovi is ((30-Den1)+Den2)) ), Denovi=<7 .
71 |
```

- **predci\_helper(Sifra,Predok)** – Ги наоѓа сите предци на лицето со дадена влезна Шифра и за секој од нив проверува дали се исполнети условите за ист пол и разлика на родендени. Поради природата на Пролог, овој предикат во Predok ќе го врати првиот предок за кој се исполнети условите, а потоа со кликање на Next се проверува дали и некој од следните предци во листата задоволуваат условите.
- **predci(Sifra, L)** – За да се соберат сите предци кои ги враќа predci\_helper во една единствена листа, се користи findall и оние лица за кој се исполнети условите се акумулираат во L.

```
73 predci_helper(Sifra,Predok):-najdi_predci(Sifra,Predci),member(Predok,Predci),
74     ist_pol(Sifra,Predok),rodendeni(Sifra,Predok).
75
76 predci(Sifra,L):-findall(Prethodnik,predci_helper(Sifra,Prethodnik),L).
```

Повикување на предикатот:



## Задача 2

### A)

Да се напише предикат во PROLOG `najbroj(X,Y)` со кој ќе се врати името X и презимето Y за бројот кој остварил комуникација со најмногу други броеви (се земаат само повиците).

Бројот кој остварил комуникација со најмногу други броеви е оној чиј збир на УНИКАТНИ телефонски броеви од појдовни и дојдовни повици е најголем. За да се најде, потребно е за секој број од базата на податоци да се најдат бровите кои ги барал, а потоа и броевите кои го барале тој број. За да не се бројат двапати оние броеви кои се наоѓаат во двете множества, користам `setof` предикат. Потоа со `findall`, се бараат се креира листа со елементи (Тел-број, Вкупно), во која треба да се најде максимум според аргументот Вкупно.

- `baral(Tel_broj,Vkupno)` – За влезниот аргумент `Tel_broj` се зема листата од појдовни Повици преку структурата `telefon`. Се враќаат телефонските броеви кои ги повикал `Tel_broj`.

```
28 baral(Tel_broj,Baran_Tel):-telefon(Tel_broj,_,_,Povici),member(povik(Baran_Tel,_),Povici).
```

- `go_barale(Tel_broj,Povikuvac)` – Се изминува базата на податоци за секој тел.број кој постои во неа. Се земаат повиците и се проверува дали бројот кој е зададен како влезен аргумент `Tel_broj` го има во листата. Доколку да, тоа значи дека моменталниот запис во databазата го барал тој број, па тогаш го враќаме бројот – повикувач. Овој предикат е потребен за да ги излиста сите броеви од база кои биле Повикувач на дадениот број `Tel_broj`.

```
30 go_barale(Tel_broj,Povikuvac):-telefon(Povikuvac,_,_,Povici),member(povik(Tel_broj,_),Povici).
```

- `e_clen(Tel_Broj, Clen)` – Овој предикат ја изминува базата и во `Clen` враќа телефонски број кој бил повикан од `Tel_broj` или го повикал `Tel_broj`.

```
32 e_clen(Tel_Broj, Clen):-go_barale(Tel_Broj,Clen);baral(Tel_Broj,Clen).
```

- **vkupno\_unikatni(Tel\_broj,Vkupno)** – Бидејќи горниот предикат ги враќа бараните и броевите кои го барале дадениот тел.број еден по еден, за да изброиме колку се, потребно е да се соберат во една листа. Тоа се постигнува со предикатот **setof**, кој враќа уникатни тел.бројеви. Како излезен аргумент се враќа должината на оваа листа – број на уникатни броеви кои го барале или биле барани од влезниот аргумент **Tel\_broj**.

```
34 vkupno_unikatni(Tel_broj,Vkupno):-setof(Clen,e_clen(Tel_broj,Clen),L),length(L,Vkupno).
```

- **maximum( L, ( MaxTelBroj , MaxPati ))** – Предикат кој го враќа бројот кој има најголем збир на уникатни броеви од појдовни и дојдовни повици.

```
37 maximum([H|O],M):-max(O,H,M).
38 max([],M,M).
39 max([(B,G)|O],(B1,G1),M):-G>G1,max(O,(B,G),M).
40 max([(B,G)|O],(B1,G1),M):- (G<G1;G=G1),max(O,(B1,G1),M).
```

- **najbroj(X,Y)** – Предикат кој генерира листа од парови (Тел-број, Вкупно), го наоѓа Максимумот во неа според вредноста Вкупно, а потоа во базата на податоци наоѓа Име и Презиме за лицето на кое припаѓа тој тел.број. Името и презимето ги враќа како излезни аргументи.

Cut операторот, спречува рекурзивна проверка назад (својствена за Пролог), односно во овој случај да не се печати false, бидејќи нема што друго да врати.

```
36 najbroj(X,Y):-findall((T,Vkupno),vkupno_unikatni(T, Vkupno),L),maximum(L,(MaxTelBroj,_)),
37 telefon(MaxTelBroj,X,Y,_),!.
```

Повикување на предикатот за дадената база на податоци:

```
X = julija,
Y = petrova
?- najbroj(X,Y)
```

## Б)

Да се напише предикат во PROLOG **omilen(X,Y)** кој за даден број X проследен на влез ќе го врати неговиот омилен број Y. Омилен број е оној со кој има остварено најголемо вкупно траење на повици (се собираат и појдовни и дојдовни). При пресметувањето на омилениот број секоја СМС порака да се разгледува како остварен повик со траење 100

(оној кој ја испраќа пораката е појдовниот број, оние до кој се испраќа пораката се дојдовни).

За да се најде омилениот број на даден влезен број, потребно е за влезниот број и за секој друг број во базата на податоци да се пресмета ЗБИР.

$$\text{ЗБИР} = \text{X-повикал-У} + \text{У-повикал-X} + \text{X-смс-У} * 100 + \text{У-смс-X} * 100.$$

Потоа да се најде максималниот збир и да се врате бројот за кој се пресметал збирот.

- `traenje_povik(X,Y,Traenje)` – предикат кој пресметува траење на повици од X кон Y. Од базата ја зема листата на повици за бројот X и проверува дали постои повик кон бројот Y. Доколку да, тогаш во Traenje ја враќа должината на повикот.

Овој предикат враќа false ако не најде повик од X кон Y. Бидејќи сакав да враќа 0 во тој случај (за да може да се изврше собирањето во главниот предикат), креирав уште еден ист предикат кој враќа 0 како трет аргумент.

Ако постои повик X кон Y, тогаш ќе се врате Траењето и потоа Cut операторот ќе спрече да се повика следниот предикат кој би вратил 0. Ако не постои повик тогаш вториот предикат ќе врате вредност 0.

\*Слично на функцијата `get_or_zero()` во Python :)

\*Предикатот е креиран со претпоставка дека сите минути за повици X кон Y ќе бидат зачувани во еден елемент `povik` (базата е креирана на тој начин). Во поинаков случај би ги собрала вредностите со `findall`.

```
46 traenje_povik(X,Y,Traenje):-telefon(X,_,_,Povici),member(povik(Y,Traenje),Povici),!.
47 traenje_povik(X,Y,0).|
```

- `dali_sms(X,Y)` – Проверува дали X пишал СМС на Y.
- `traenje_sms(X,Y,Traenje)` – За влезни броеви X и Y, ги наоѓа сите СМС во базата и тој број го множи со 100, па го враќа во Traenje. Овде нема потреба да се креира уште еден предикат (како кај повиците) кој би враќал 0 ако нема СМС, бидејќи `findall` во тој случај враќа празна листа. Должината на празна листа е 0, па  $0*100 = 0$ .

```
49 dali_sms(X,Y):-sms(X,SMS), member(Y,SMS).|
50
51 traenje_sms(X,Y,Traenje):-findall(_,dali_sms(X,Y),L),length(L,Len),Traenje is (Len*100).
52
```

- `traenje(X,Y,Total)` – Предикат за пресметување на формулата ЗБИР запишана на почеток од задачата, која ја имплементира главната логика. Овој предикат е креиран да „не фрла ерор“, односно да не паѓа доколку некој од не постои повик или смс во некоја насока, туку тогаш траењето е 0.

```

53 traenje(X,Y,Total):-telefon(Y,_,_,_),
54   traenje_povik(X,Y,Traenje1),
55   traenje_povik(Y,X,Traenje2),
56   traenje_sms(X,Y,Traenje3),
57   traenje_sms(Y,X,Traenje4),
58   Total is (Traenje1+Traenje2+Traenje3+Traenje4),write(Total).

```

- **omilen(Broj,Omilen)** – Главниот предикат кој за влезен број Broj, генерира листа од елементи-парови (Tel\_broj,Vkupno), а потоа бара максимум според аргументот Вкупно и го враќа Tel\_broj како Омилен.

```

63 omilen(Broj,Omilen):-findall((X,T),traenje(Broj,X,T),L),maximum(L,(Omilen,_)).

```

Повикување на предикатот за дадената база на податоци:

<pre> Y = 222222 false ?- omilen(111111,Y) </pre>	<pre> Y = 161616 false ?- omilen(666666,Y) </pre>
---	---

## Задача 3

### A)

Напишете предикат во PROLOG **izbroj\_lokacija(Lok,Br)** кој за локација Lok која се задава на влез ќе пресмета колку пати таа локација била почетна или крајна за некоја услуга.

За да се изброи колку пати една локација била почетна и крајна, потребно е да се влезе во услугите на секој клиент од базата на податоци и таму да се провери дали има записи за локацијата.

- **counter(Lok,Br,Uslugi)** – за дадена локација Lok и влезна листа од Услуги, враќа колку пати се појавува локацијата во таа листа (Br). Информациите од елементите од листата се пристапуваат со директно пристапување преку обликот на структурата [usluga(Pocetna,Krajna,\_,\_)|L].  
Предикатот е рекурзивен, така што основниот случај е кога сме стигнале до крај на листата па бројачот се иницијализира на 0.  
Другиот повик е кога дадената локација е Почетна или Крајна во моменталниот запис кој го проверуваме. Тогаш се повикува рекурзивно предикатот за останатите елементи, а при враќање бројачот ќе се зголеми за 1 бидејќи условот

е исполнет. Овде има CUT оператор бидејќи во следниот услов (линија 25) нема логичка проверка, а не би требало да влегува во него доколку е исполнет условот во линија 24 што би се случило поради backtracking на Пролог.

- `po_klient(Lok,Br)` – За секој клиент од базата на податоци пребројува колку пати се среќава дадената Локација во неговите Услуги со предикатот `counter`. Се пристапува до листата со услуги преку структурата на Клиент и потоа таа се предава на предикатот `counter`.
- `izbroj_lokacija(Lok,Br)` – Главниот предикат кој ги собира пребројувањата за дадена локација за СЕКОЈ клиент во една листа. Потоа ги собира елементите во таа листа и збирот го враќа на излез. CUT операторот е поставен за да не враќа `false` после враќање на одговорот.

```

23 counter(Lok,0,[]):-!.
24 counter(Lok,Br,[usluga(Pocetna,Krajna,_,_)|L]):- (Pocetna==Lok;Krajna==Lok),counter(Lok,Br1,L),Br is Br1+1,!.
25 counter(Lok,Br,[usluga(Pocetna,Krajna,_,_)|L]):-counter(Lok,Br1,L),Br is Br1.
26
27 po_klient(Lok,Br):-klient(_,_,Uslugi), counter(Lok,Br,Uslugi).
28
29 izbroj_lokacija(Lok,Br):-findall(OdKlient,po_klient(Lok,OdKlient),L),sum_elementi(L,Br),!.
30
31 sum_elementi([X],X).
32 sum_elementi([X|L],Br):-sum_elementi(L,Br1), Br is X+Br1.
33

```

Повикување на предикатот за дадената база на податоци:

<b>Br = 5</b>	<b>Br = 7</b>
?- izbroj_lokacija(a,Br).	?- izbroj_lokacija(f,Br).

## Б)

Напишете предикат во PROLOG `najmnogu_kilometri(X,Y)` кој во X и Y ќе ги враќа името и презимето на клиентот кој има поминато најмногу километри со такси компанијата.

За да се најде таква информација, потребно е за секој клиент да се пресмета колку километри има изминато преку листата од неговите услуги. Но, во базата имаме информација за километри само за директно растојание меѓу две локации, па затоа потребно е да се имплементира предикат кој пресметува најкратко растојание меѓу две локации. (Кога таксистите возат помеѓу две локации ја поминуваат патеката која има минимална должина од сите можни патеки помеѓу локациите).

Откако ќе ја имаме таа информација за секој клиент, го бараме клиентот со максимум изминати километри и за него се враќаат име и презиме.



- **presmetaj\_rastojanie(A,B,Pominati,Km)** – предикат кој за влезни локации Od и Do го враќа растојание меѓу нив во променливата Km. Во базата имаме податоци само за едната насока, па затоа кога се бара растојание треба да се провери дали постои од A до B ИЛИ од B до A → основниот случај.

Доколку нема директен пат потребно е да се пресмета индиректно растојание со сите измеѓу локации. Важно е да се чува листа на изминати патеки за да не се минува повторно низ исти патишта што може да воде до бесконечен циклус или подолги патишта.

Пример: (g,d) -> (g,f,b,d)   НО, ако не ја чуваме оваа помошна листа тогаш откако ќе го најде (g,f), ќе почне да пребарува патеки (f,\_) што Пролог ќе го унифицира со (f,g) и повторно ќе почне да пребарува за истото. Затоа откако наоѓаме соодветна патека за поминување се проверува дали е член на листата на Поминати. Ако не е, се додава (при тоа се додаваат записи и во двете насоки!) и потоа рекурзијата продолжува.

На крај се собираат километрите од патеката Od->Preku и од рекурзивниот повик Preku->Do.

Овој предикат ги враќа сите можни растојание меѓу две локации.

- **najkratko\_rastojanie(A,B,MinKm)** – За да се најде најкраткото растојание во сите можни растојанија од локација A до B, се повикува findall на претходниот предикат и потоа се бара најмалиот број километри со предикатот najdi\_minimum (не е објаснет бидејќи го имаме работено на аудиториски).

```

46 presmetaj_rastojanie(Od,Do,_,Km):- (rastojanie(Od,Do,Km) ; rastojanie(Do,Od,Km)).
47 presmetaj_rastojanie(Od,Do,Pominati,Km):-
48     (rastojanie(Od,Preku,Km1);rastojanie(Preku,Od,Km1)),
49     not(member((Od,Preku),Pominati)),
50     append([(Od,Preku),(Preku,Od)],Pominati,NovoPominati),
51     presmetaj_rastojanie(Preku,Do,NovoPominati,Km2), Km is (Km1+Km2).
52
53
54 najkratko_rastojanie(A,B,MinKm):-findall(Km,presmetaj_rastojanie(A,B,[],Km),L) ,najdi_minimum(L,MinKm).
55
56 najdi_minimum([X|_],M) :- minimum(0,X,M),!.
57 minimum([X|_],Y,M) :- X>=Y, minimum(0,Y,M).
58 minimum([X|_],Y,M) :- X<Y, minimum(0,X,M).
59 minimum([ ],M,M).

```

- **kolku\_klient(Uslugi>Total)** – предикат кој за дадена листа на Услуги на еден Клиент враќа вкупен број на изминати километри. За таа цел, за секој запис usluga, се зема почетната и крајната локација и се пресметува најкраткото растојание меѓу нив. Потоа рекурзивно се повикува истата пресметка и за останатите записи во листата. Резултатот од рекурзијата се собира со резултатот од моменталниот запис и се враќа на излез. Основниот случај на рекурзијата е кога листата е празна, па се прави иницијализација на километрите на 0.

- **klienti(Id,Total)** – Предикат кој за даден клиент со ИД, го пребарува записот во базата на податоци, ја зема листата со Услуги и потоа го повикува предикатот за пресметување на Вкупно километри за тој клиент.
- **najmnogu\_kilometri(Ime,Prezime)** – Предикат кој со findall ги зема сите записи (КлиентИД, ВкупноКМ) за секој клиент од базата на податоци и ги става во листа. Потоа бара кој запис има најмногу километри со предикатот maximum, кој е модифициран да прима листа од сложени записи, да споредува според Километри, а потоа да врате ИД на клиентот со најмногу КМ и бројот на КМ. За да се вратат името и презимето на тој клиент мора да се пристапи до структурата klient со ИДто кое го дознаваме од maximum.

```

61 %za sekoja usluga na klientot presmetaj kolku km e taa i potoa soberi gi site
62 kolku_klient([],0).
63 kolku_klient([usluga(Pocetna,Krajna,_,_) |Uslugi], Total):-najkratko_rastojanie(Pocetna,Krajna,Ras),
64                                     kolku_klient(Uslugi,SubTotal),Total is (SubTotal+Ras).
65
66 klienti(Id,Total):-klient(Id,_,Uslugi),kolku_klient(Uslugi,Total).
67
68 najmnogu_kilometri(Ime,Prezime):-findall((KlientId,TotalKM) , klienti(KlientId,TotalKM), L), maximum(L,(MaxKlientID,_))
69                                     ,klient(MaxKlientID,Ime,Prezime,_).
70
71 maximum([H|_],M):-max(0,H,M).
72 max([],M,M).
73 max([(B,G)|_],(B1,G1),M):-G>G1,max(0,(B,G),M).
74 max([(B,G)|_],(B1,G1),M):-G<G1;G=G1,max(0,(B1,G1),M).

```

Повикување на предикатот за дадената база на податоци:

```

?- najmnogu_kilometri(Ime,Prezime)
Ime = vasil,
Prezime = vasiliev
?-

```

**B)**

Напишете предикат во PROLOG najmnogu\_zarabotil(X) со кој ќе го најдете бројот на такси возилото X кое заработило најмногу во текот на месец декември 2015 година.

Потребно е да се најдат сите броеви на такси возила кои вршат услуги. Бидејќи ги немаме како директни записи во базата на податоци, потребно е да се пристапат преку листите со услугите на клиентите. Потоа, за секој број на такси возило да се најдат услугите кои ги има направено во Декември, 2015. За тие услуги потребно е да се пресмета колку километри се изминати и да се помножи тој број со цената за километар. Откако ќе ги имаме и овие пресметки, потоа за секое такси возило потребно е да се соберат тие броеви за да се добие Сума колку заработил. На крај, се бара чија сума е најголема од такси возилата и се враќа бројот на такси возило.

- **najdi\_broj\_vozilo(Broj\_Vozilo)** – предикат за пристапување на податокот за број на возило кој е достапен во листата на Услуги на секој од клиентите.
- **unikatni\_broj\_vozilo(L)** – Предикат кој враќа листа од уникатни вредности за број на возило. За таа цел се користи предикатот **setof**. Пресметките би се извршиле и да не се уникатни вредностите, но со ова се постигнува оптимизација!
- **uslugi\_dekemvri(Broj\_Vozilo,Naplata)** – За дадено број возило, пресметува колку заработил од услугите во декември 2015. Овој предикат враќа по еден запис за секој клиент. Се пристапува до листата на Услуги на Клиентот, потоа се изминува секоја од услугите и се филтрираат само оние кои се направени од Broj\_Vozilo во Декември. За да се постигне ова при земање на елементите од листата, се прави унификација со **datum(\_,12,2015)**, Broj\_Vozilo. Останатите информации од Услуга ни се исто така потребни за да се пресмета колку километри биле изминати и колку е крајната цена на таа услуга со множење на бројот на километри и цената. Бројот на километри се наоѓа преку предикатот **najkratko\_rastojanie** кој е објаснет во претходните задачи.
- **suma\_uslugi\_dekemvri(Broj\_Vozilo,Suma)** – Бидејќи претходниот предикат враќа пресметка за секој клиент, ако сакаме да дознаеме колку заработило такси возилото потребно е да се соберат сите тие записи. Затоа се користи **findall**, се зема пресметаната Naplata, а потоа на сите наплати им се пресметува сума со предикатот **sum\_elementi**, кој е објаснет во претходните задачи.  
Со овој предикат имаме информација за секој од такси возилата колку заработил.
- **najmnogu\_zarabotil(MaxBrojVozilo)** – Овој предикат ги зема сите записи од претходниот предикат и потоа го бара такси возилото кое остварило најголема заработка. Соодветно, за него го враќа Бројот на такси возилото на излез.

```

209 %V) Za sekoj broj na taxi vozilo koe moze da se najde vo bazata na podatoci, da se zemat uslugite koi gi ima napraveno vo dekemvri,2015
210 % Da se presmeta kolku kilometri izminal za taa dadena usluga i da se pomnozi so cenata na uslugata.
211
212 najdi_broj_vozilo(Broj_Vozilo):-klient(_,_,Uslugi),member(usluga(_,_,Broj_Vozilo),Uslugi).
213 unikatni_broj_vozilo(L):-setof(Broj_Vozilo,najdi_broj_vozilo(Broj_Vozilo),L).
214
215 uslugi_dekemvri(Broj_Vozilo,Naplata):-klient(_,_,Uslugi),member(usluga(Od,Do,Cena,datum(_,12,2015),Broj_Vozilo),Uslugi),
216     najkratko_rastojanie(Od,Do,Km), Naplata is (Km*Cena).
217
218 suma_uslugi_dekemvri(Broj_Vozilo,Suma):-findall(Naplata,uslugi_dekemvri(Broj_Vozilo,Naplata),L),sum_elementi(L,Suma),!.
219
220 unikatni_vozila_sumi(Broj_Vozilo,Suma):-unikatni_broj_vozilo(L),member(Broj_Vozilo,L),suma_uslugi_dekemvri(Broj_Vozilo,Suma).
221 najmnogu_zarabotil(MaxBrojVozilo):-findall((Broj_Vozilo,Suma),unikatni_vozila_sumi(Broj_Vozilo,Suma),L),maximum(L,(MaxBrojVozilo,_)),!.
222

```

Повикување на предикатот за дадената база на податоци:

X = 34

?- najmnogu\_zarabotil(X)