Документација за домашна 1

Задача 1 непарен палиндром

Предикати

neparen_palindrom – Програмата е дизајнирана да одговара на овој предикат.

neparen_palindrom([_]). - Доколку аргуметот е листа која што се состои само од еден член тогаш таа е непраен палиндром.

neparen_palindrom([Glava|Ostatok]):-

izvadi_posleden(Ostatok,Posleden,ListOut),

Glava = Posleden,

neparen_palindrom(ListOut).

Во случај кога аргументот пренесен на **neparen_palindrom** е листа со повеќе елементи тогаш подели ја листата на **Glava** и **Ostatok** потоа со помош на предикатот **izvadi_posleden Ostatok** подели го на **Posleden** (последниот елемент од листата) и **ListOut** (ги содржи останатите елементи). Понатаму осигурај се дека првиот елемент на влезната листа е еднаков со последниот (**Glava = Posleden).** Потоа провери дали остнатата листа (без првиот и последниот елемент (**ListOut**)) е непарен палиндром **neparen_palindrom(ListOut).**

izvadi_posleden – предикат кој е наменет да се користи така што како прв аргумент се предеава листа, а како втор аргумент се предава променлива во која ќе се зачува последниот елемент од листата предадена како прв, и како трет аргумент променлива во која ќе се зачува листата предадена како прв аргумент без последниот елемент

izvadi_posleden([X],X,[]).

Доколку првиот аргумент е листа од само еден елемент тогаш тој елемент е последен а преостаната листа е празна.

izvadi_posleden([Glava|Ostatok],Posleden,ListOut):-

izvadi_posleden(Ostatok,Posleden,ListOutRec),

ListOut=[Glava|ListOutRec].

Во случај кога првиот аргумент е листа со повеќе елементи тогаш подели ја листата на Glava и Ostatok и продолжи да го бараш последнито елемент од Ostatokot рекурзивно **izvadi_posleden(Ostatok,Posleden,ListOutRec),** дополнително додаи го првиот елемент (Glava) на почеток од излезната листа **ListOut=[Glava|ListOutRec].**

Забелешка: Не е потребна дополнителна проверка дали **L** листата внесена како аргумент на **neparen_palindrom(L)** има непарен број на елементи бидеќи во секоја итерација на **neparen_palindrom** вадиме по два елементи првиот и последниот и со тоа всушност се врши

проверка на парноста, т.е. во случај листата да е парен палиндром кратеќи ја по два елементи ќе се доведеме до ситуација да се бара вистинитоста на **neparen_palindrom([])** и како резултат ќе се врати False бидеќи неможе да се направи унификација

Задача 2: нај подниза

Предикати:

%naj_podniza(L,N,X). X e podlista so dolzina N koja sto se pojavuva najmnogu pati vo L

naj_podniza(InputL,Dolzina,Maks):-

findall(L1,(podniza(InputL,L1),dolzina(L1,Dolzina)),L),

naj_pojavuvana(L,Maks),!.

Со помош на **findall** ги наоѓаме сите подлисти на влезната листа **InputL** со должина **Dolzina** и ги зачувуваме во листа **L**. Потоа со помош на **naj_pojavuvana** го наоѓаме елементот (подлистата) кој се појавува најмногу пати во листата од подлисти.

%podniza(L1,L2). L2 e podnizna na L1

podniza([X|L1],[X|L2]):-podnizaPocetok(L1,L2).

podniza([_|L1],L2):-podniza(L1,L2).

Предикатот **podniza(L1,L2)**, враќа True доколку листата **L2** е подлиста на **L1**. Итерирај ја првата листа се додека не најдеш елемент кој е ист со првиот елемент од втората листа, тогаш со предикатот **podnizaPocetok** провери дали остатокот од листата започнува со втората листа, односно дали втората листа е подлиста која се наоѓа на почеток од остатокот од првата листа

%podnizaPocetok(L1,L2). L2 e podniza koja sto se naoga na pocetok na L1,

podnizaPocetok(_,[]).

podnizaPocetok([X|L1],[X|L2]):-podnizaPocetok(L1,L2).

Предикатот **podnizaPocetok** е помошен предикат на предикатот **podniza** работи така што ги крати двете листи доколку започнуваат со ист член се додека не се испразни втората листа.

%dolzina(L,N) N e brojot na elementi vo L

dolzina([],0).

dolzina([_|L],N):-dolzina(L,M), N is M+1.

Предикатот **dolzina** враќа должина на листа.

%naj_pojavuvana(L,Maks) Maks e eleemntot koj sto najmnogu se pojavuva vo L

naj_pojavuvana([X|L],Maks):-izbroiPojavuvana(X,[X|L],N),naj_pojavuvana(L,[X|L],X,N,Maks).

Предикатот **naj_pojavuvana** го наоѓа елементот кој што се појавува најмногу пати во дадената листа. Најпрвин ги сетираме помошните променливи **tempX** и **tempN** (го чуваат тековно најдениот елемент со најмногу појавувања) на првиот член во листата и бројот на негови појавувања во неа, предикатот **izbroiPojavuvana** се користи за го добиеме бројот на појавувања, потоа ја повикуваме преоптеретената верзија на предикатот **naj_pojavuvana** со 5 аргументи.

%naj_pojavuvana(L,L,tempX,tempN,X). X e element koj sto se pojavuva najmnogu pati vo L.

% tempX i tempN se pomosnini promenlvi koj go cuvaat tekovno najdeniot element so najmnogu pojavuvana inicijalno se setirani na prviot element I negoviot broj na pojavuvana

naj_pojavuvana([],L,MaksX,MaksN,MaksX).

naj_pojavuvana([X|O],L,MaksX,MaksN,Out):-

izbroiPojavuvana(X,L,N),N>MaksN,

naj_pojavuvana(O,L,X,N,Out).

naj_pojavuvana([X|O],L,MaksX,MaksN,Out):-

izbroiPojavuvana(X,L,N),N=<MaksN,

naj_pojavuvana(O,L,MaksX,MaksN,Out).

Ја изминуваме листата елемент по елемент и проверуваме дали моменталниот елемент се појавува повеќе пати во листата од тековно најдениот елемент со најмногу појавувања. Доколку тоа е исполнето тој станува тековен елемент со најмногу појавувања. Кога ќе се измине цела листа елементот со најмногу појавувања е тековниот елемент со нај појавувања. За броење на појавувања на елемент во листа се користи предикатот **izbroiPojavuvana.**

%izbroiPojavuvana(X,L,N). N e broj na pojavuvana na X vo listata L.

izbroiPojavuvana(_,[],0).

izbroiPojavuvana(X,[X|L],N):-izbroiPojavuvana(X,L,M), N is M+1.

izbroiPojavuvana(X,[_|L],N):-izbroiPojavuvana(X,L,N).

Предикатот **izbroiPojavuvana** брои колко пати се појавува дадениот елемент во дадената листа. Изминувај ја листата елемент по елемент доколку моменталниот елемент е еднаков на даденито зголеми го бројачот. Кога ќе ја изминеш листата иницијализирај го бројачот на 0

Задача 3: помал-поголем-помал pattern

Предикати:

proveri([A,B]):-B>A.

proveri([A,B,C|Ostatok]):-proveri_uslov3([A,B,C|Ostatok]).

proveri(List) – Програмата е дизајнирана да дава одговори на овој предикат

Доколку влезниот аргумент е листа која се состои од точно два елемнти тогаш таа го задоволува барањето ако вториот елемент е поголем од првиот. Доколку влезниот аргумент е листа која што се состои од три или повеќе елемнти тогаш провери дали важи условот 3 од барањата на задачата. Проверката на условот 3 се прави со предикатот **proveri_uslov3(List)**

proveri_uslov3([_]).

proveri_uslov3([A,B]):-B>A.

proveri_uslov3([A,B,C|Ostatok]):-B>A, C<B, proveri_uslov3([C|Ostatok]).

proveri_uslov3(List) – Предикат кој што го проверува третиот услов од барањата на задачата односно дали првиот елемент е помал од вториот, вториот е поголем од третиот, третиот помал од четвртиот и тн.

proveri_uslov3([_]). – Доколку влезниот аргумент е листа која што се состои од точно еден елемент значи дека листата го задоволува услов 3

proveri_uslov3([A,B]) :- B>A.– Во слусчај влезниот аргумент да е листа која што се состои од точна два елементи тогаш листата го задоволува услов 3 ако вториот елемент во листата е поголем од првиот.

proveri_uslov3([A,B,C|Ostatok]):-B>A, C<B, proveri_uslov3([C|Ostatok]).

Доколку листата се состои од три или повеќе елементи тогаш провери дали вториот елемент е поголем од првиот и дали третиот елемент е помал од вториот. Потоа рекурзивно провери дали листата без првите два елементи го задоволува услов 3.

Забелешка: И покрај тоа што предикатот **proveri_uslov3(List)** дава True доколку листата има само еден елемент, не е прекршени условот 1 бидеќи тоа се користи само како граничен случај во рекурзијата, односно условот 1 се проверува така што за предикатот **proveri(List)** постои унификација само доколку проследениот аргумент е листа од два или повеќе елементи. Па

така предикатот **proveri_uslov3(List)** ќе биде повикан само доколку оригиналната листа содржи барем 3 елементи.

Задача 4 пермутации

%permutacii(L1,L2). L2 e lista od site permutacii na L1

permutacii(L1,L2):-setof(P,permutacija(P,L1),L2).

Со користење на **setof** со педикатот **permutacija** ги наоќаме сите пермутации на влезната листа.

%permutacija(L1,L2) True ako L1 e permutacija na L2

permutacija([],[]).

permutacija([X|L1],L2):-izvadiElem(L2,X,L3), permutacija(L1,L3).

Предикатот **permutacija** проверува дали првата листа е пермутација на втората, така што ја изминуваме првата листа и за секој елемент го вадиме истиот од втората листа доколку го содржи. Вадењето на елемент од втората листа се прави со предикатот **izvadiElem**

%izvadiElem(L1,X,L2). od listata L1 isvadi go elementot X a ostatokot zapisi go vo L2

izvadiElem([X|L],X,L).

izvadiElem([X|L1],Y,[X|L2]):-izvadiElem(L1,Y,L2).

Предикатот **izvadiElem** прима листа и елемент и ја враќа листата без тој елемент, доколку елементот не се пронајде се враќа False. Ова се постигнува така што ја изминуваме листата се додека не го најдеме бараниот елемент а во излезната листа ги додаваме сите елементи освен најдениот.

Забелешка: се користи **setof** наместо **findal** поради тоа што во ситуација **permutacii([1,2,1],L)** **findall** враќа L= [[1, 2, 1], [1, 1, 2], [2, 1, 1], [2, 1, 1], [1, 1, 2], [1, 2, 1]] а **setof** враќа

L=[[1, 1, 2], [1, 2, 1], [2, 1, 1]].

Задача 5 бинарни аритметички операции

%convert(L,X). X e decimalniot zapis na brojot L koj e vo binaren zapis vo format na lista

convert(L,Out) :- convert(L,0,Out).

convert([],Tmp,Tmp).

convert([X|Rest],Tmp,Out) :-

NewTmp is (Tmp * 2) + X,

convert(Rest,NewTmp,Out).

Предикатот **Convert** ја претвора влезната листа која што представува бинарен број во декадена форма. Предикатот работи така што најпво се повикува преоптоварената верзија која содржи дополнителна променлива за зачувување на привремен резултат чија вредност иницијално е 0. Потоа ја изминуваме влезната листа елемент по елемент и привремениот резултат го зголемуваме двојно и го додаваме моменталниот елемент. Кога ќе ја изменеме цела листа привремениот резултат го враќаме на излез.

%convertR(X,L). L e binaren zapis vo format na lista na decimalniot broj X

convertR(X,L):-convertR(X,[],L).

convertR(0,L,L):-!.

convertR(X,L,Out):-

Ostatok is X mod 2,

Kolicnik is X // 2,

convertR(Kolicnik,[Ostatok|L],Out).

Предикатот **convertR** прима број и враќа листа која го претставува бинарниот запис на бројот.

Конверзијата се одвива така што бројот целобројно го делиме со 2 се додека не добиеме 0, но меѓутоа при секоја итерација го зачувуваме остатокот при делење со два во листа акомулатор, кога ќе стигнеме до 0 излезната листа се изедначува со акомулираната. (забелешка: коритиме листа акомулатор за излезната листа ја добиеме во правилниот редослед).

%sobiranje(L1,L2,R):- R e zbir od L1 i L2 kade i L1 i L2 i R se vo binaren zapis

sobiranje(L1,L2,R):-

convert(L1,X1),

convert(L2,X2),

R_dec is X1 + X2,

```
   convertR(R_dec,R).


%odzemanje(L1,L2,R):- R e razlika od L1 i L2 kade i L1 i L2 i R se vo binaren zapis
odzemanje(L1,L2,R):-
   convert(L1,X1),
   convert(L2,X2),
   R_dec is X1 - X2,
   (R_dec>0, convertR(R_dec,R),!);
         R=[0].


%mnozenje(L1,L2,R):- R e proizvod od L1 i L2 kade i L1 i L2 i R se vo binaren zapis
mnozenje(L1,L2,R):-
   convert(L1,X1),
   convert(L2,X2),
   R_dec is X1 * X2,
   convertR(R_dec,R).


%delenje(L1,L2,R):- R e celobroen kolicnik od L1 i L2 kade i L1 i L2 i R se vo binaren zapis
delenje(L1,L2,R):-
   convert(L1,X1),
   convert(L2,X2),
   R_dec is X1 // X2,
   convertR(R_dec,R).
```

Сите предикати **sobiranje, odzemanje, mnozenje, delenje,** работат на сличен принцип првин се конвертираат влезните бинарни броеви во декадни и потоа се врши бараната операцијата и потоа резултатот се конвертира ва бинарна листа.

Задача 6 Матрици

```
%presmetaj(M,R) vlez e matrica M a izlez e R dobieno kako M*M_transponirana
%R_ij=X_i1*X_j1 + X_i2*X_j2 + X_i3*X_j3 + ...
```

%kade R_ij e pozicija i,j vo izleznata matrica R, X_ij e pozicija i,j vo vleznata matrica X

presmetaj(M,R):-presmetaj(M,M,1,R),!.

%presmetaj(M,M,I,R). M e vleznata matrica, R e izleznata I e iterator po redici.

presmetaj(_,[],_,[]).

presmetaj(M,[X|L],I,[Y|L2]):-

presmetajRed(M,X,I,1,Y),

I_new is I+1,

presmetaj(M,L,I_new,L2).

Се итерира матрицата ред по ред и се повикува предикатот **presmetajRed** за секој ред.

%presmetajRed(M,L,I,J,R). M-vlezna matrica, L e I-tiot red vo matricata, a R e I-tiot red od rezultatnata matrica

% J e iterator po koloni

presmetajRed(_,[],_,_,[]).

presmetajRed(M,[_|L1],I,J,[Y|L2]):-

presmetajKelija(M,M,I,J,1,Y),

J_new is J+1,

presmetajRed(M,L1,I,J_new,L2).

Влезни параметри во **presmetajRed** се оригиналната матрица, ред од матрицата, и реден број на редот, а излез е ред од резултатната матрица,
Се итерира секој елемент од редот и за секој елемент се повикува предикатот **presmetajKelija.**

%presmetajKelija(M,M,I,J,Iterator,R). M e vleznata matrica,R kelija na pozicija (I,J) vo rezultantnata matrica

presmetajKelija(_,[],_,_,_,0).

presmetajKelija(M,[_|L],I,J,Iterator,R):-

clen_pozicija_matrica(M,I,Iterator,X1),

clen_pozicija_matrica(M,J,Iterator,X2),

P is X1 * X2,

Iterator_new is Iterator + 1,

presmetajKelija(M,L,I,J,Iterator_new,R2),

R is R2 + P.

Влезни параметри се две копии од влезната матрица една во која се чува матрицата, а другата копија за итерација, позицијата на ќелијата која се пресметува (I, J)=(ред, колона), излез е вредноста на резултатната ќелија.

Во првата итерација зе земаат вредностите на ќелиите со позиција (i, 1) и (j,1), се пресметува нивниот производ и се додава на сумата која што всушност е резултатот, во вотрата итерација се земаат ќелиите на позиција (i, 2) и (j,2)) и се пресметува нивниот прозивот, во следна итерација на (i, 3) (j,3), итн се до (i, N) (j,N)) каде N е големината на матрицата. Односно

*R_ij=X_i1*X_j1 + X_i2*X_j2 + X_i3*X_j3 + ...*

kade R_ij e vrednosta na pozicija i,j vo izleznata matrica R, X_ij e vrednosta na pozicija i,j vo vleznata matrica X.

%clen_pozicija_matrica(M,I,J,X) X e elementot na pozicija I,J vo matricata X

clen_pozicija_matrica([X|_],1,J,Y):-clen_pozicija(X,J,Y).

clen_pozicija_matrica([_|L],I,J,X):-I_new is I-1, clen_pozicija_matrica(L,I_new,J,X).

%clen_pozicija(L,N,X). X e elementot na N-ta pozicija vo L.

clen_pozicija([X|_],1,X).

clen_pozicija([_|L],I,X):-I_new is I-1, clen_pozicija(L,I_new,X).

За земање на елемент на дадена позиција во матрицата се користи предикатот **clen_pozicija_matrica** кој итерира низ матрицата ред по ред се додека не стигне до бараниот ред па потоа го повикува предикатот **clen_pozicija** кој итерира елемент по елемент се додека не стигне до параната позиција и го враќа елементот на таа позиција.

Задача 7 сортирање подлисти

transform(L1,L2):-izbrisiDuplikati(L1,L3),sortiraj_podlisti(L3,L2),!.

Предикатот **transform** функционира така што првин се отсрануваат дупликатите со предикатот **izbrisiDuplikati** во влезната листа а потоа се сортира со **sortiraj_podlisti**.

%sortiraj_podlisti(L1,L2), L2 e podredena L1 kade L1 e sostavena od podlisti

sortiraj_podlisti([],[]):-!.

sortiraj_podlisti(L1,[Max|L2]):-

izvadiNajgolem(L1,Max,Rest),

sortiraj_podlisti(Rest,L2),!.

sortiraj_podlisti – Најди го најголемиот елемент стави го на почеток на излезната листа, сортирај го остатокот.

%izvadiNajgolem(L,Max,Rest) Max e najgolemiot element vo L, a Rest e L bez Max

izvadiNajgolem(L,Max,Rest):- max(L,Max),izvadiElem(L,Max,Rest).

izvadiNajgolem – Најди го најголемиот елемент потоа извади го од листата, за вадење на елемент се користи предикатот **izvadiElem** дефиниран во задача 4-пермутации.

%max(L,X). X e najgolemiot element vo L

max([X],X).

max([X|L1],Maks):-

max(L1,MaksR),

pogolem(X,MaksR), Maks = X.

max([X|L1],Maks):-

max(L1,MaksR),

not(pogolem(X,MaksR)), Maks = MaksR.

Max – Итерирај ја листата елемент по елемент и провери дали моменталниот елемент е поголем од најголемиот елемент во остатокот (**MaksR),** доколку е постави го моменталниот елемент како најголем, во спротивно постави го **MaksR.** За споредба дали некој елемент е поголем се користи предикатот **pogolem** (бидеќи споредуваме листи).

%pogolem(L1,L2) True ako L1 ima poveke elementi od L2,

%vo slucaj da imaat isti elementi se proveruva dali prviot elem od L1 e pogolem od prviot od L2

%ako i prvite elementi se ednakvi se proveruvaat vtorite i tn.

pogolem(L1,L2):-

dolzina(L1,N1),

dolzina(L2,N2),

N1>N2.

pogolem(L1,L2):-

dolzina(L1,N),

dolzina(L2,N),

pogolemiElementi(L1,L2).

Pogolem – доколку првата листа има повеќе елементи од втората врати True, доколку имаат ист број на елементи провери го предикатот **pogolemiElementi.**

%pogolemiElementi(L1,L2) True ako prviot elem od L1 e pogolem od prviot od L2

%ako prvite elementi se ednakvi se proveruvaat vtorite i tn.

pogolemiElementi([X|_],[Y|_]):- X>Y.

pogolemiElementi([X|L1],[X|L2]):- pogolemiElementi(L1,L2).

pogolemiElementi – Доколку првиот член на првата листа е поголем од првиот член на втората врати True,а доколку се еднакви провери ги вторите и тн.

%izbrisiDuplikati(L1,L2) L2 e lista od unikatnite elementi od L1.

izbrisiDuplikati([],[]).

izbrisiDuplikati([X|L1],L2):-clen(X,L1), izbrisiDuplikati(L1,L2).

izbrisiDuplikati([X|L1],[X|L2]):-not(clen(X,L1)), izbrisiDuplikati(L1,L2).

izbrisiDuplikati – Изминувај ја листата елемнт по елемнт, доколку моменталниот елемент е член во остатокот тоа значи дека е дупликат и не го додавај во излезната листа.

%clen(X,L) True ako X e element na L.

clen(X,[X|_]).

clen(X,[_|L]):-clen(X,L).

Задача 8 бриши секое второ

brisi_sekoe_vtoro(L,R):-najdi_unikati(L,Unikati),brisi_sekoe_vtoro(L,Unikati,R).

brisi_sekoe_vtoro(L,R) – Најпрво најди ги уникатните елементи во L листа која содржи и подлисти, потоа повикај ја преоптоварената верзија на предикатот brisi_sekoe_vtoro каде ја ја предадеш и листата со уникатни елементи.

brisi_sekoe_vtoro(L,[],L).

brisi_sekoe_vtoro(L,[X|L2],R):-

brisi_sekoe_vtoro_X(L,X,0,R2,_),

brisi_sekoe_vtoro(R2,L2,R).

Итерирај ја листата со уникати и за секој елемент повикај го предикатот **brisi_sekoe_vtoro_X,** резултатната листа од **brisi_sekoe_vtoro_X** пратија како влез (листа од која се брише) за следната итерација**.** Кога ќе стигнеме до крај од листата со уникати тогаш моменталната листа од која се брише е всушност резултантана листа.

%brisi_sekoe_vtoro_X(L,X,Flag,R,FlagOut) R e dobieno taka sto od L e izbrisano sekoe vtoro pojavuvane na X.

%L moze da sodrzi i podlisti, Flag=0- preskokni sledno X, Flag=1 brisi sledno X.

%FlagOut go ima istoto znacene so Flag samo sto e izlezen parametar, se koristi vo vgnezdenite rekurzii.

brisi_sekoe_vtoro_X([],_,Flag,[],Flag).

brisi_sekoe_vtoro_X([X|L],X,0,[X|R],FlagOut):-brisi_sekoe_vtoro_X(L,X,1,R,FlagOut).

brisi_sekoe_vtoro_X([X|L],X,1,R,FlagOut):-brisi_sekoe_vtoro_X(L,X,0,R,FlagOut).

brisi_sekoe_vtoro_X([G|L],X,Flag,[G|R],FlagOut):-not(lista(G)), G\==X, brisi_sekoe_vtoro_X(L,X,Flag,R,FlagOut).

brisi_sekoe_vtoro_X([G|L],X,Flag,[R2|R],FlagOut):-lista(G),

brisi_sekoe_vtoro_X(G,X,Flag,R2,FlagOutR),

brisi_sekoe_vtoro_X(L,X,FlagOutR,R,FlagOut).

brisi_sekoe_vtoro_X – Изминувај ја влезната листа елемент по елемент доколку момнеталниот елемент е ист со елементот што треба да се брише и **Flag** има вредност 0, не го бриши елементот (додади го во излезната листа) и постави го **Flag** на 1.

Во случај моменталниот елемент да е ист со елементот за бришење и **Flag=1,** тогаш избриши го елементот (не го додавај во излезната листа), и постави го **Flag=0.**
Во случај моменталниот елемент да не е ист со елементот што се брише и не е листа тогаш не го бриши.
Доколку моменталниот елемент е листа тогаш повикај **brisi_sekoe_vtoro_X** каде моменталниот елемент ќе го пратиш како влезна листа од која ќе се брише секое второ појавување на X, дополнително прати ја и моменталната вредност на **Flag,** резултатната листа на вгнездената рекурзија додади ја како елемент во резултатната листа на надворешната рекурзија, дополнително вредноста на **Flag** постави ја на излезната вредност **FlagOutR** од вгнездената рекурзија и продолжи со изминување на листата елемент по елмент.
Кога ќе ја изминеме цела листа значи дека сме завршиле и сега на излезната **FlagOut** додели и ја вредноста на моменталниот Flag.

%lista(X) True ako X e lista

lista([]).

lista([_|_]).

%najdi_unikati(L,L2) L2 e lista koja gi sodrzi samo unikatnite edinecni elementi od L kade L ima podlisti

```
najdi_unikati(L,R):-najdi_unikati(L,[],R).

najdi_unikati([],Acc,Acc).

najdi_unikati([X|L1],Acc,R):-not(lista(X)), not(clen(X,Acc)), najdi_unikati(L1,[X|Acc],R).

najdi_unikati([X|L1],Acc,R):-not(lista(X)), clen(X,Acc), najdi_unikati(L1,Acc,R).

najdi_unikati([X|L1],Acc,R):-lista(X),
   najdi_unikati(X,Acc,R2),
   najdi_unikati(L1,R2,R).
```

najdi_unikati – Итерирај ја влезната листа елемент по елемент доколку момнеталниот елемент не е листа и доколку не е член во листата акомулатор, додади го во акомулаторот,
доколку момнеталниот елемент не е листа но е член во листата акомулатор не го додавај,
Доколку момнеталниот елемент е листа повикај го предикатот **najdi_unikati** каде како влезна листа ќе го предадеш моменталниот елемент, а на влезниот акомулатор ќе го предадеш моменталниот акомулатор, резултатната листа од вгнездената рекурзија е новиот акомулатор, продолжи со итерирање се додека не ја испразниш цела листа тогаш резултатната листа е акомулаторот.