5. Reprezentarea sistemelor cu IA bazate pe cunoştinţe în SWI-Prolog.

5.1 Baze de cunoştinţe dinamice

În Prolog programul se actualizează automat în timpul execuției sale prin utilizarea predicatelor predefinite:

assert, asserta, assertz, retract, retractall.

Adăugarea de fapte şi reguli în baza de cunoştinţe se face cu predicatele ***assert, asserta, assertz***, iar eliminarea se face cu predicatul **retract**.

Predicatul ***asserta(Clauza)*** permite adăugarea clauzei **Clauza** la începutul bazei de cunoştinţe Prolog.

Predicatele ***assertz(Clauza)*** şi ***assert(Clauza)*** au o comportare similară cu ***asserta***, cu excepția faptului că argumentul ***Clauza*** este adăugat la sfârşitul bazei de cunoştinţe.

Predicatul ***retract(Clauza)*** caută prima clauză din baza de cunoştinţe care se potriveşte cu *Clauza*, unifică cele două clauze, după care clauza găsită este ştearsă din baza de cunoştinţe.

Exemplu de utilizare:

Considerând baza de cunoştinţe de mai jos:

```
:- op(200, xfx, [este, are]).
:- op(200, xf, inoata).
broasca are piele.
broasca este animal.
broasca inoată.
```

Pentru a putea adăuga noi fapte în program trebuie mai întâi să precizăm că, în mod dinamic, putem adăuga noi fapte. Acest lucru se realizează prin comanda:

:-dynamic(este/2).

În acest moment putem adăuga de la consolă noi fapte de tipul *este*. De exemplu, dacă adăugăm faptul:

```
?- assert(papagalul este animal).
```

la interogarea:

```
?- papagalul este X.
```

Sistemul va răspunde cu:

```
X = animal.
```

Se pot adăuga în program şi clauze noi astfel:

```
25 ?- assert(student(popescu)).
26 ?- assert(student(georgescu)).
27 ?- assert(student(ionescu)).
```

La interogarea:

```
28 ?- student(X).
```

Sistemul va răspunde cu:

```
X = popescu ;
X = georgescu ;
X = ionescu.
```

Se pot elimina clauze din baza de cunoştințe folosind predicatul *retract*, astfel:

```
41 ?- retract(student(georgescu)).
```

La interogarea:

```
42 ?- student(X).
```

Sistemul va răspunde cu:

```
X = popescu ;
X = ionescu.
```

5.2 Operatori definiți de programator

Un exemplu de operator definit de programator în SWI-Prolog este următorul:

:-op(Precedenta, Tip, Nume).

Precedența (prioritatea) determină ordinea de efectuare a operațiilor într-o expresie cu diverşi operatori. Precedența este un număr cu valori între 0 şi 1200. Precedența pentru = este, de exemplu, 700, precedența pentru + este 500, iar pentru * este 400. De exemplu, 2+3*4 este interpretat în SWI-Prolog ca 2+(3*4) deoarece + are precedența mai mare decât *.

Tipul este un atom ce specifică tipul şi asociativitatea operatorului. Tipul operatorului se precizează folosind una din următoarele forme standard:

- Infixat, cu trei forme: xfx xfy yfx
- Prefixat, cu două forme: fx fy
- Postfixat, cu două forme: xf yf

în care ***f*** reprezintă operatorul (numele operatorului), iar ***x*** şi ***y*** operanzii săi.

În tabelul 5.1 prezentăm operatorii standard utilizați de limbajul SWI-Prolog.

Tabelul 5.1 *Operatori standard utilizaţi de limbajul SWI-Prolog*

Precedență (prioritate)	Tip (specificator)	Nume operatori
1200	xfx	:-
1200	fx	:- ?-
1100	xfy	;
1050	xfy	->
1000	xfy	,
900	fy	not
700	xfx	**=, is, < , > , =< , >= , ==, =\= , \== , =:=**
500	yfx	+ -
400	yfx	* / // mod
200	xfy	^
200	fy	-

Asociativitatea indică ordinea de efectuare a operațiilor într-o secvență de operații care au aceeaşi precedență. De exemplu, 2+3+4 va fi interpretat ca (2+3)+4 (asociativitate la stânga) sau 2+(3+4) (asociativitate la dreapta). Semnificațiile lui **x** şi **y** în stabilirea tipului operatorului sunt următoarele:

- **x** reprezintă un argument (operand) cu precedență strict mai mică decât cea a functorului (operatorului) **f**

$$\text{precedența}(\ \mathbf{x}\ ) < \text{precedența}(\ \mathbf{f}\ )$$

- **y** reprezintă un argument (operand) cu precedență mai mică sau egală cu cea a functorului (operandului) **f**

$$\text{precedența}(\ \mathbf{y}\ ) \leq \text{precedența}(\ \mathbf{f}\ )$$

Aceste reguli ajută la eliminarea ambiguității operatorilor multipli cu aceeaşi precedență.

De exemplu, operatorul predefinit în Prolog **-** (minus) este definit din punct de vedere al tipului ca **yfx**, ceea ce înseamnă că structura a - b - c este

interpretată ca (a - b) - c şi nu ca a - (b - c). Dacă acest operator ar fi fost definit ca **xfy**, atunci interpretarea structurii a - b - c ar fi fost a - (b - c).

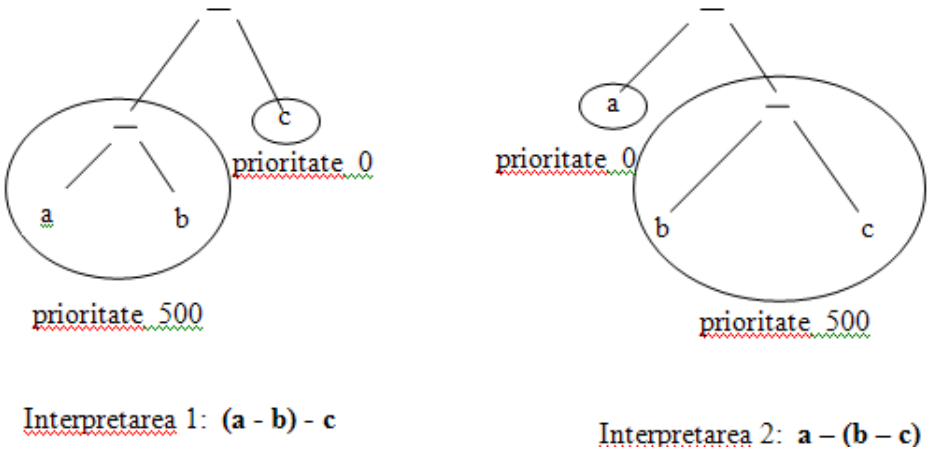

Figura 5.1 *Două interpretări ale expresiei **a – b – c**, presupunând că **' – '** are prioritatea 500. Dacă **' – '** este de tipul **yfx,** atunci interpretarea 2 este invalidă pentru că prioritatea lui **b – c** nu este mai mică decât prioritatea lui **'– '**.*

Numele operatorului poate fi orice atom Prolog care nu este deja definit în Prolog. Se poate folosi şi o listă de atomi, dacă se definesc mai mulți operatori cu aceeaşi precedență şi acelaşi tip.

Exemplu de utilizare:

```
:- op(200, xfx, [este, are]).
:- op(200, xf, inoata).
broasca are piele.
broasca este animal.
broasca sare.
```

Testare:

```
17 ?- Cine are piele.
Cine = broasca.

18 ?- Cine sare.
Cine = broasca.
```

5.3 Predicate de interacțiune cu utilizatorul

- ***Read*** **-** limbajul SWI-Prolog permite interacțiunea cu utilizatorul prin intermediul predicatului *read*.
  Sintaxa sa este:

read(X).

unde X este o variabilă, urmată de caracterele pe care programul trebuie să le citească. Read(X) are ca rezultat unificarea variabilei X cu şirul de caractere de intrare.
Astfel, dacă scriem:

```
1 ?- read(X).
|: laborator.
```

Prolog va satisface *read(X)*, unificând variabila X cu constanta *laborator*, rezultând:

```
X = laborator.
```

Atunci când se urmăreşte citirea mai multor cuvinte într-o singură variabilă se procedează astfel:

```
8 ?- read(X).
|: 'bazele inteligentei artificiale'.
```

Testare:

```
X = 'bazele inteligentei artificiale'.
```

- ***Get* şi *put*** – predicate utilizate pentru scrierea şi citirea caracterelor**.**

SWI-Prolog are predicate predefinite pentru scrierea şi citirea a câte unui caracter. Predicatul *put* va afişa la consolă un singur caracter, însă, argumentul său trebuie să fie un întreg care reprezintă caracterul în cod ASCII. Aşadar, în Prolog, pentru scrierea textului, ar trebui să apelăm:

```
11 ?- put(112), put(114), put(111), put(108),
      put(111), put(103).
prolog
true.
```

Pentru acest exemplu, este de preferat să folosim predicatul *write('prolog')* prezentat în capitolele precedente. Dacă folosim ca argument la predicatul write un mesaj scris între ghilimele, atunci vor fi afişate chiar valorile întregi care reprezintă fiecare caracter în cod ASCII.

```
12 ?- write("prolog").
[112, 114, 111, 108, 111, 103]
true.
```

Pentru citirea unui caracter folosim predicatul *get* care citeşte, din cele introduse, doar primul caracter. Poate fi introdus orice caracter şi atunci, la citire, nu mai trebuie să încheiem, după ce am introdus caracterul de citit, cu punct (.). Punctul poate fi chiar caracterul de citit.

De exemplu, predicatul *citesc* definit în continuare, citeşte un caracter şi afişează valoarea sa corespunzătoare în codul ASCII:

```
citesc :- write('Introduceti un caracter:'), get(C),
          nl, write('Valoarea caracterului '), put(C),
          write(' in cod ASCII este: '), write(C).
```

Răspunsul sistemului va fi:

```
15 ?- citesc.
Introduceti un caracter:s
Valoarea caracterului s in cod ASCII este: 115
true.
```

- **Predicate predefinite pentru prelucrare şiruri de caractere**

Limbajul SWI-Prolog furnizează predicate predefinite pentru procesare stringuri. Dintre acestea, cele mai utilizate sunt:

string_concat(+String1, +String2, -String3) – realizează concatenarea a două stringuri (**String1** şi **String2**) în al treilea (**String3**).

Exemplu de utilizare:

```
14 ?- string_concat(laborator, ' test', Mesaj).
Mesaj = "laborator test".
```

string_to_list(+String, -ASCII) sau **string_to_list(-String, +Ascii)** – realizează trecerea de la atomi la lista care conține numere întregi în cod ASCII. Cel puțin unul din cele două argumente trebuie să fie instanțiat.

Exemple de utilizare:

```
18 ?- string_to_list('student',Lista).
Lista = [115, 116, 117, 100, 101, 110, 116].

19 ?- string_to_list(String,[115, 116, 117, 100, 101, 110, 116]).
String = "student".
```

string_length(+String, -Lungime) – determină lungimea şirului de caractere.

Exemplu de utilizare:

```
21 ?- string_length(student,Lungime).
Lungime = 7.
```

string_to_atom(+String, -Atom) sau **string_to_atom(-String, +Atom)** – realizează trecerea de la string la atom sau invers.

Exemple de utilizare:

```
27 ?- string_to_atom(evaluare,Atom).
Atom = evaluare.

28 ?- string_to_atom(String, 22.15).
String = "22.15".
```

sub_string(+String, +Start, +Lungime, -Rest, -Substring) – determină un substring din stringul **String**. Primul argument (**String**) este şirul de caractere din care extragem un subşir, **Start** este un întreg pozitiv care păstrează poziția de la care selectăm subşirul, al treilea argument (**Lungime**) păstrează lungimea subşirului, **Rest** este un întreg pozitiv care spune câte poziții mai sunt până la sfârşitul şirului inițial, iar ultimul argument (**Substring**) reprezintă chiar subşirul selectat.

Exemple de utilizare:

```
35 ?- sub_string('studentii sunt la curs', 0, 15, Rest, Valoare).
Rest = 7,
Valoare = "studentii sunt ".

36 ?- sub_string('studentii sunt la curs', 10, 5, Rest, Valoare).
Rest = 7,
Valoare = "sunt ".
```

Menționăm că şi în cazul acestui predicat, ca şi la ***string_concat***, argumentele de intrare sunt atomi. Numai argumentul de ieşire reprezintă un string. Prin urmare, numai primul argument trebuie să fie întotdeauna instanțiat, celelalte putând fi calculate.

convert_time(+Timp, -Valoare) – transformă data şi ora curentă din *float* în şir de caractere. Primul argument (**Timp**) poate fi obținut cu ajutorul predicatului *get_time*. În acest fel obținem data şi ora curente.

Exemplu de utilizare:

```
 5 ?- get_time(Timp), convert_time(Timp,Data),
      nl, write('Data este '), write(Data).
Data este Fri Sep 11 14:53:55 2009
Timp = 1.25267e+009,
Data = "Fri Sep 11 14:53:55 2009".
```

Pentru separarea elementelor care alcătuiesc data şi ora curentă se foloseşte predicatul ***convert_time*** astfel:

```
6 ?- get_time(Timp), convert_time(Timp, An, Luna,
     Zi, Ora, Minute, Secunde, Milisecunde).
 Timp = 1.25267e+009,
An = 2009,
Luna = 9,
Zi = 11,
Ora = 14,
Minute = 57,
Secunde = 21,
Milisecunde = 671.
```

- ***Consult* şi *reconsult***

Înainte de a utiliza un program SWI-Prolog acesta trebuie încărcat în memorie. Această operație este realizată cu ajutorul comenzii:

consult('NumeFisier').

Cu acest predicat avem posibilitatea să folosim un program Prolog dintr-un fişier. Predicatul *consult('NumeFisier')* reuşeşte întotdeauna, având drept rezultat salvarea clauzelor care sunt în „*NumeFişier*" în memoria internă a calculatorului, astfel încât ele pot fi folosite de Prolog pentru a trage concluzii.

Predicatul:

reconsult('NumeFisier').

reîncarcă un fişier compilat Prolog.

5.4 Operații cu fişiere

Intrarea şi ieşirea pentru date (altele decât cele asociate interogării programului) sunt făcute prin predicate predefinite. Orice sursă sau destinație de date este numită în Prolog stream (canal de intrare sau de ieşire). Cele mai utilizate predicate pentru citirea şi scrierea datelor sunt, read şi write. Ambele au un singur argument şi folosesc canalul curent de intrare, respectiv ieşire. Terminalul utilizatorului este tratat ca un fişier, fiind denumit **user**.

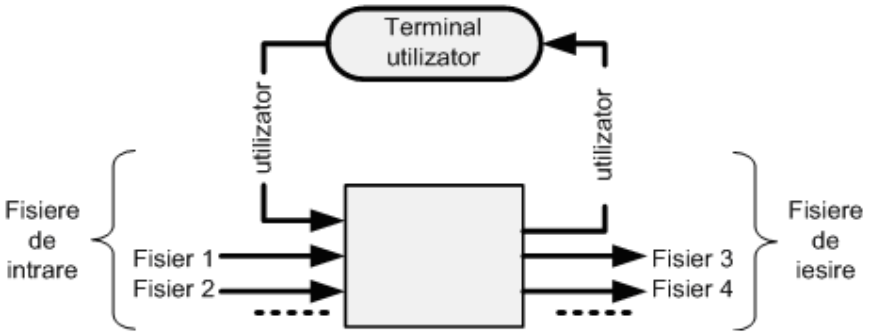

Figura 5.2 *Modelul de comunicație program Prolog – fișiere de intrare-ieșire*

Terminalele (canalele) curente predefinite sunt tastatura şi ecranul. Există un flux curent de date de intrare şi un flux curent de date de ieşire. Comutarea între fluxurile de date se poate face cu predicatele:

- **see(Fisier)** - fişierul dat ca argument devine fluxul curent de intrare (fişier de intrare curent). Fişierul **Fişier** este deschis pentru citire, iar pointer-ul de fişier este poziționat la începutul lui.
- **seen** - închide fişierul de intrare curent (fluxul curent de intrare) şi stabileşte tastatura ca şi canal de intrare curent.
- **tell(Fisier)** – deschide fişierul **Fisier** pentru scriere şi îl face fişier de ieşire curent (fişierul **Fisier** devine fluxul curent de ieşire).
- **append(Fisier)** – deschide fişierul **Fisier** pentru scriere în mod adăugare (pointer-ul este mutat la sfârşitul fişierului) şi îl face fişier de ieşire curent (fişierul **Fisier** devine fluxul curent de ieşire).
- **told** – închide fişierul de ieşire curent stabilind ecranul ca stream de ieşire (închide fluxul curent de ieşire).

Notă:

- Predicatul predefinit read(X) citeşte în variabila X următorul termen din fişierul de intrare (vezi punctul 5.3: **Interacțiunea cu utilizatorul: predicatul predefinit *read***). Termenul citit trebuie să se termine, în interiorul fişierului, cu caracterul punct.
- În Prolog, există o constantă specială numită ***end_of_file*** care este returnată atunci când s-a ajuns la sfârşitul fişierului (s-au citit toate datele din fişier).
- Pentru formatarea ieşirii există două proceduri:
  - *nl* - scrie un caracter sfârşit de linie,

- tab(N) – scrie N caractere blank (adaugă N spații față de poziția la care este situat canalul de ieşire curent).

5.5 Probleme rezolvate

1. Program interactiv.

În cele ce urmează prezentăm un exemplu de program în Prolog pentru interacțiunea cu utilizatorul. Se construieşte o bază de cunoştințe în care se memorează țările şi capitalele acestora. Utilizatorul, prin intermediul tastaturii, poate solicita informații despre o țară prin introducerea numelui acesteia.

Codul sursă SWI-Prolog este următorul:

```
capitala(romania, bucuresti).
capitala(bulgaria, sofia).
capitala(anglia, londra).
capitala(china, beijing).
capitala(australia, canberra)
capitala(japonia, tokio).

start  :-
write('Capitala carei țări vă interesează?'),nl,
write('Scrieți numele țării, litere mici, urmat de punct.'),
nl,
read(Stat),
capitala(Stat, Capitala),
write('Capitala este:'),
write(Capitala),nl.
```

Rezultate obținute la testare:

```
12 ?- start.
Capitala carei tari va intereseaza?
Scrieti numele tarii, litere mici, urmat de punct.
|: anglia.
```

Răspunsul sistemului, la interogarea de mai sus, este:

```
Capitala este:londra
true.
```

2. În cele ce urmează prezentăm un program Prolog ce implementează o rețea semantică pentru diagrama din figura 5.3. În această aplicație vom folosi, pentru construirea bazei de cunoştințe, operatorii definiți de

utilizator. Cunoştinţele sunt reprezentate ca o colecţie de concepte (nodurile reţelei) între care există diverse relaţii (arcurile reţelei). De exemplu, relaţiile de tipul *este un* sau *este o* permit moştenirea proprietăţilor. Pentru scrierea aplicaţiei în limbaj natural vom utiliza operatorii definiţi de programator.

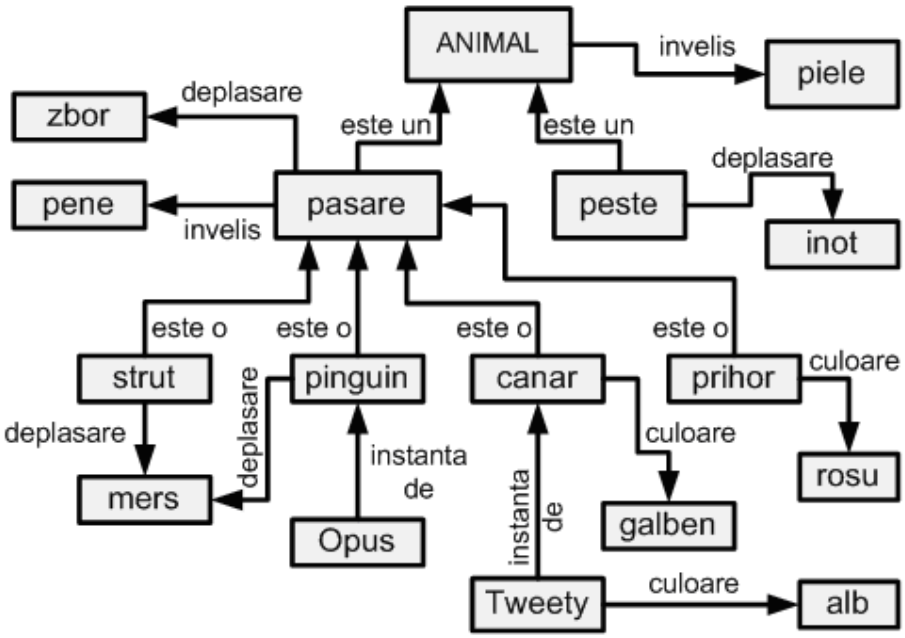

Figura 5.3 *Structură arborescentă pentru clasificare animale*

Codul sursă SWI-Prolog este următorul:

```
% Date: 13/04/2009
% Operatori definiti de utilizator
:-op(500, xfx, este_o).
:-op(500, xfx, este_un).
:-op(600, xfx, un).
:-op(600, xfx, o).
:-op(500, fx, este).
:-op(500, xfx, este_instanta_de).
:-op(500, xfx, este_acoperita_de).
:-op(500, xfx, este_acoperit_de).
:-op(500, xfx, se_deplaseaza_prin).
:-op(500, xfx, are_culoarea).
% :-op(400, xf,?).
% :-op(400,xf,[:-,?-]).
:-op(650, fx, ce).
:-op(500, xfx, culoare).
:-op(500, xfx, invelis).
:-op(500, xfx, deplasare).
:-op(500, xfx, are).

strutul este_o pasare.
pinguinul este_o pasare.
canarul este_o pasare.
prihorul este_o pasare.
pasarea este_un animal.
pestele este_un animal.
opus este_instanta_de pinguin.
tweety este_instanta_de canar.
canarul are_culoarea galbena.
prihorul are_culoarea rosie.
tweety are_culoarea alba.
pinguinul se_deplaseaza_prin mers.
strutul se_deplaseaza_prin mers.
pasarea se_deplaseaza_prin zbor.
pestele se_deplaseaza_prin inot.
pasarea este_acoperita_de pene.
animalul este_acoperit_de piele.

articuleaza(pasare, pasarea).
articuleaza(peste, pestele).
articuleaza(animal, animalul).
articuleaza(canar, canarul).
articuleaza(pinguin, pinguinul).
```

```
mostenire(A,B):-
                A este_un B, write(A), write(' este un '),
                write(B), write('.'),nl.
mostenire(A,B):-
                A este_o B, write(A), write(' este o '),
                write(B), write('.'),nl.
mostenire(A,B):-
                A este_instanta_de B, write(A),
                write(' este instanta  de '), write(B),
                write('.'),nl.
mostenire(A,C):-
                mostenire(A,B), articuleaza(B,B1),
                mostenire(B1,C).
mostenire(A,C):-
                mostenire(A,B), articuleaza(B,B1),
                mostenire(B1,C).

(este A un B ?)  :- mostenire(A,B).
(este A o B ?)   :- mostenire(A,B).

gaseste_culoare(A):-
                    A are_culoarea B, write(A),
                    write(' are culoarea '), write(B),
                    write('.'),nl.
gaseste_culoare(A):-
                    mostenire(A,B),articuleaza(B,B1),
                    gaseste_culoare(B1).
gaseste_invelis(A):-
                    A este_acoperit_de B, write(A),
                    write(' este acoperit de '), write(B),
                    write('.'),nl.
gaseste_invelis(A):-
                    A este_acoperita_de B, write(A),
                    write(' este acoperita de '), write(B),
                    write('.'),nl.
gaseste_invelis(A):-
                    mostenire(A,B), articuleaza(B,B1),
                    gaseste_invelis(B1).
```

```
gaseste_deplasare(A):-
                    A se_deplaseaza_prin B, write(A),
                    write(' se deplaseaza prin '), write(B),
                    write('.'),nl.
gaseste_deplasare(A):-
                    mostenire(A,B),articuleaza(B,B1),
                    gaseste_deplasare(B1).

(ce este A ?) :- mostenire(A,X).
(ce culoare are A ?) :- gaseste_culoare(A).
(ce invelis are A ?) :- gaseste_invelis(A).
(ce deplasare are A ?) :- gaseste_deplasare(A).
```

Rezultate obținute pe parcursul testării:

```
53 ?- ce este pasarea? .
```

Răspunsul sistemului, la interogarea de mai sus, este:

```
pasarea este un animal.
true
```

```
54 ?- este tweety un animal? .
```

Răspunsul sistemului este:

```
tweety este instanta de canar.
canarul este o pasare.
pasarea este un animal.
true
```

```
57 ?- ce culoare are canarul? .
```

Răspunsul sistemului este:

```
canarul are culoarea galbena.
true
```

```
62 ?- ce invelis are strutul? .
```

Răspunsul sistemului este:

```
strutul este o pasare.
pasarea este acoperita de pene.
true
```

```
67 ?- ce deplasare are prihorul? .
```

Răspunsul sistemului este:

```
prihorul este o pasare.
pasarea se deplaseaza prin zbor.
true
```

3. Construire meniu.

Un exemplu de program în SWI-Prolog ce afişează utilizatorului un meniu şi execută comanda selectată.

Codul sursă este prezentat în cele ce urmează:

```
%baza de cunostinte
capitala(china, beijing).
capitala(bulgaria, sofia).
capitala(romania, bucuresti).
capitala(rusia, moscova).

interpretare(49,china).
interpretare(50,bulgaria).
interpretare(51,romania).
interpretare(52,rusia).

%proceduri de interactiune cu utilizatorul
start:- afisare_meniu,
        citeste_din_meniu(Tara),
        capitala(Tara,Oras),
        nl, write('Capitala tarii '),
        write(Tara),
        write(' este orasul '),
        write(Oras), nl.


afisare_meniu :-
          write('Capitala carei tari va intereseaza?'),
          nl,
          write('1 China'), nl,
          write('2 Bulgaria'), nl,
          write('3 Romania'), nl,
          write('4 Rusia'), nl,
          write('Scrieti un numar de la 1 la 4 -- ').
citeste_din_meniu(Tara) :-
          get(Cod_ASCII), %citeste un caracter
          interpretare(Cod_ASCII,Tara).
```

Un posibil dialog cu programul arată astfel:

```
18 ?- start.
Capitala carei tari va intereseaza?
1 China
2 Bulgaria
3 Romania
4 Rusia
Scrieti un numar de la 1 la 4 -- 4
Capitala tarii rusia este orasul moscova
true.
```

4. Accesare fişiere.

Un exemplu de program în Prolog care citeşte numerele dintr-un fişier text (*fişier.txt*) şi scrie în fişierul *pare.txt* numerele care sunt pare, iar în fişierul *impare.txt* numerele care sunt impare.

O posibilă implementare în SWI-Prolog următoarea:

```
testare :- see('c:/prolog/fisier.txt'), citesc([]), seen.

citesc(L) :- read(N), N \= end_of_file,
             append(L, [N], Rezultat), citesc(Rezultat).
citesc(L) :- separare(L, Pare, Impare),
             pare(Pare),
             impare(Impare).

separare([], [], []).
separare([P|R1], [P|R2], L2) :- Rest is P mod 2, Rest = 0,
                                separare(R1, R2, L2).
separare([P|R1], L2, [P|R2]) :- Rest is P mod 2, Rest = 1,
                                separare(R1, L2, R2).

pare(L) :- tell('c:/prolog/pare.txt'), afisare(L), told.
impare(L):-tell('c:/prolog/impare.txt'),afisare(L),told.

afisare([]).
afisare([Prim|Rest]) :- write(Prim), write('.'), nl,
                        afisare(Rest).
```

La testare:

```
8 ?- testare.
```

în fişierul *pare.txt* vor fi scrise numerele pare, iar în fişierul *impare.txt* numerele impare.

5.6 Probleme propuse

1. Se vor studia problemele rezolvate (problemele prezentate pe parcursul acestui laborator), încercând găsirea altor posibilități de soluționare a acestora. Utilizați şi alte scopuri (interogări) pentru a testa definițiile predicatelor introduse. Se atrage atenția asupra faptului că toate cunoştințele din acest capitol vor fi necesare şi în derularea celorlalte capitole.
2. Se vor rezolva următoarele probleme propuse şi se va urmări execuția lor corectă.
   - Realizați un program („invata_capitale") Prolog capabil să memoreze sau să adauge noi informații în baza de cunoştințe pe măsură ce interacționează cu utilizatorul. Adăugarea de noi informații se va face prin utilizarea predicatului ***assert*** prezentat, în acest capitol, la punctul 5.1. Programul solicită utilizatorului, prin intermediul tastaturii, numele țării despre care vrea să afle informații. Dacă țara introdusă nu există în baza de cunoştințe, atunci programul îi cere utilizatorului să introducă numele capitalei acelei țări. Noua informație este memorată în memoria sistemului de calcul.
   - Definiți un operator (op) astfel încât *place(student, facultate)* sa poată fi scris: *student place facultate*.
   - Propuneți o definiție pentru operatorii (‚este', ‚uneia', ‚dintre'), astfel încât să poată fi scrise clauze de genul: *diana este secretara uneia dintre admiteri*. În urma interogărilor, sistemul va trebui să răspundă astfel:

```
34 ?- diana este Ce.
Ce = secretara uneia dintre admiteri.

35 ?- Cine este secretara uneia dintre admiteri.
Cine = diana.
```

   - Scrieți o interogare Prolog care să elimine din aplicația rezolvată 3 toate aparițiile predicatului ***este_o***. Verificați corectitudinea execuției programului.
   - Scrieți o interogare Prolog care să elimine din aplicația rezolvată 3 numai apariția predicatului ***este_o*** în care avem termenul *struț*. Verificați corectitudinea execuției programului.
   - Modificați problema rezolvată 2 astfel încât faptele din baza de cunoştințe să poată fi scrise sub forma: *capitala*

nume_tara *este* nume_capitala. Exemplu: *capitala României este Bucureşti*.

- Utilizând fişierul *fişier.txt* din problema rezolvată 5, scrieți un program Prolog care să scrie în fişierul *sortat.txt* şirul de numere ordonat crescător.
- Utilizând fişierul *fişier.txt* din problema rezolvată 5, scrieți un program Prolog care să scrie în fişierul *prime.txt* doar numerele care sunt prime.
- Avem la dispoziție 5 culori: alb, galben, roşu, verde, albastru. Să se precizeze toate drapelele tricolore care se pot forma cu aceste culori, ştiind că trebuie respectate două reguli:
    - orice drapel are culoarea din mijloc galben sau verde.
    - cele trei culori de pe drapel sunt distincte.