



Structuri compuse in Prolog

Ruxandra Stoean
<http://inf.ucv.ro/~rstoean>
ruxandra.stoean@inf.ucv.ro



Ce sunt structurile?

- Cea mai mare parte din informația pe care vrem să o reprezentăm într-un program este compusă, adică ea constă din entități care au mai multe atribute diferite
 - De exemplu, entitatea *persoană* poate avea mai multe atribute precum *vârstă*, *înălțime*, *greutate* etc.
- În programele realizate până acum, am utilizat numai fapte și reguli care foloseau structuri de date simple.
 - Argumentele predicatelor folosite au fost atomi sau numere.
- Aceste date simple pot însă fi combinate pentru a construi tipuri de date complexe numite **structuri**.

Ce sunt structurile?



- O structură constă dintr-un functor și un număr fix de argumente:

`structura(arg1, arg2, ..., argn)`

- Fiind un **termen**, structura poate să apară în cadrul unei clauze oriunde poate apărea o variabilă sau o constantă.
- Precum toți ceilalți termeni folosiți anterior, nici structurile nu trebuie declarate
 - Le folosim direct în cadrul programului, acolo unde avem nevoie de ele.



Ce sunt structurile?

- Deși arată precum predicatele, nu funcționează ca acestea.
 - Predicatele sunt *relatii*, iar structurile sunt *obiecte*.
 - Diferența dintre cele două este realizată de către Prolog după locul în care apar într-o clauză.
- Structurile nu apar niciodată singure, ci doar ca argumente pentru predicate.

Exemplu de utilizare a unor structuri

```
student(adrian, prezente(8), proiect(bun)).  
student(marius, prezente(2), proiect(copiat)).  
student(andreea, prezente(9), proiect(bun)).  
student(ovidiu, prezente(4), proiect(slab)).
```

- Daca vrem sa aflam care sunt studentii care pot sa ia o nota peste 7, putem folosi interogarea:
 - ? – student(X, prezente(Y), proiect(bun)), Y > 6, writeln(X), fail.
adrian
andreea



Exemplu de utilizare a unor structuri

student(adrian, prezente(8), proiect(bun)).

student(marius, prezente(2), proiect(copiat)).

student(andreea, prezente(9), proiect(bun)).

student(ovidiu, prezente(4), proiect(slab)).

- Dacă vrem să aflăm care sunt studenții care nu vor intra în examen, putem folosi interogarea:

- ? – student(X, prezente(_Y), proiect(copiat)), writeln(X), fail.
marius

Underline-ul indică faptul că nu suntem interesați de valoarea cu care este unificat acest câmp.



Alt exemplu de utilizare a unor structuri

- O structura poate avea si mai mult de un singur argument (cum a fost cazul in exemplul precedent).
- Exemplu:

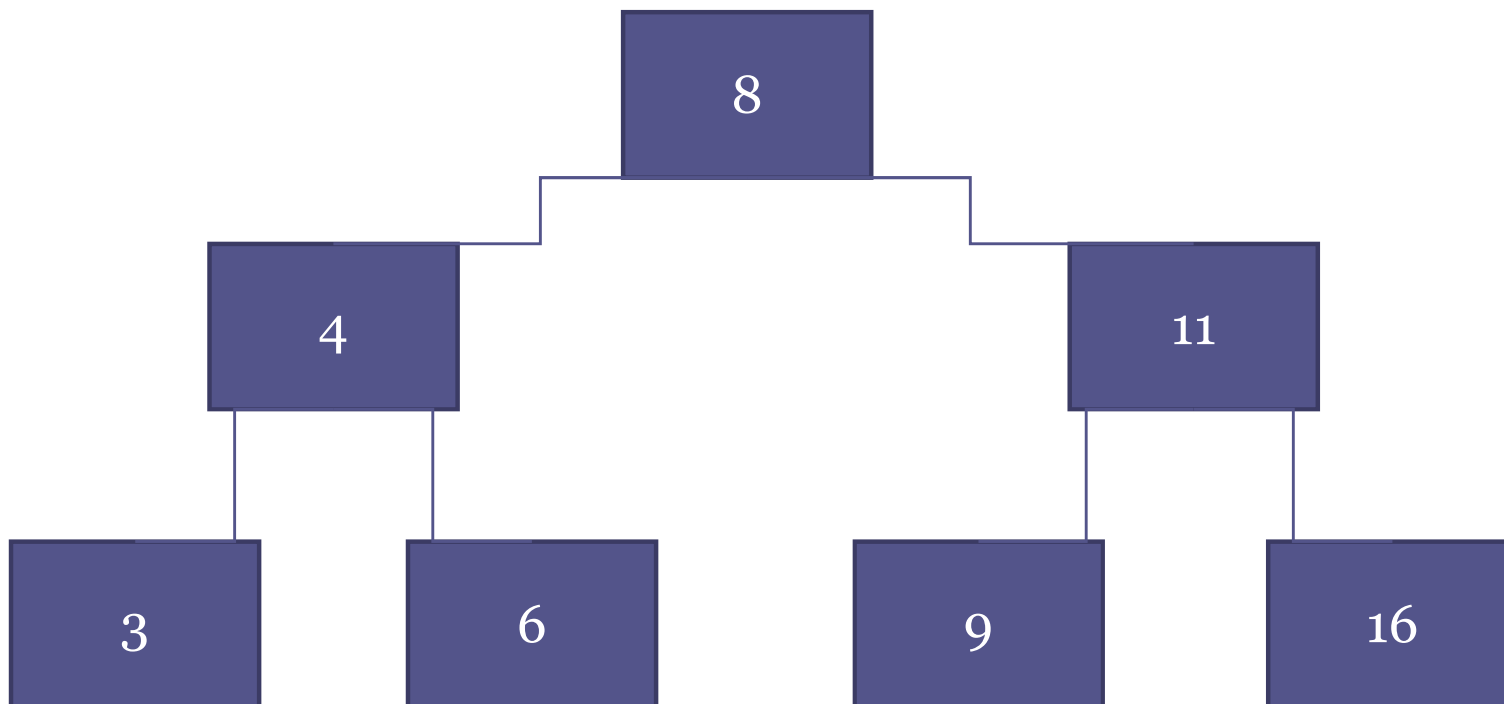
are(ionut, calculator(asus, '3 Ghz', 'RAM 1 GB')).

are(ovidiu, calculator(hp, '2.7 Ghz', 'RAM 512 MB')).

are(ovidiu, calculator(dell, '2.4 Ghz', 'RAM 512 MB')).

Arbori binari

- Un arbore binar are proprietatea că pentru un nod părinte:
 - fiecare nod aflat în partea stângă a sa are o valoare numerică mai mică decât a sa și
 - fiecare nod aflat în partea dreaptă a nodului părinte are o valoare mai mare decât a sa.

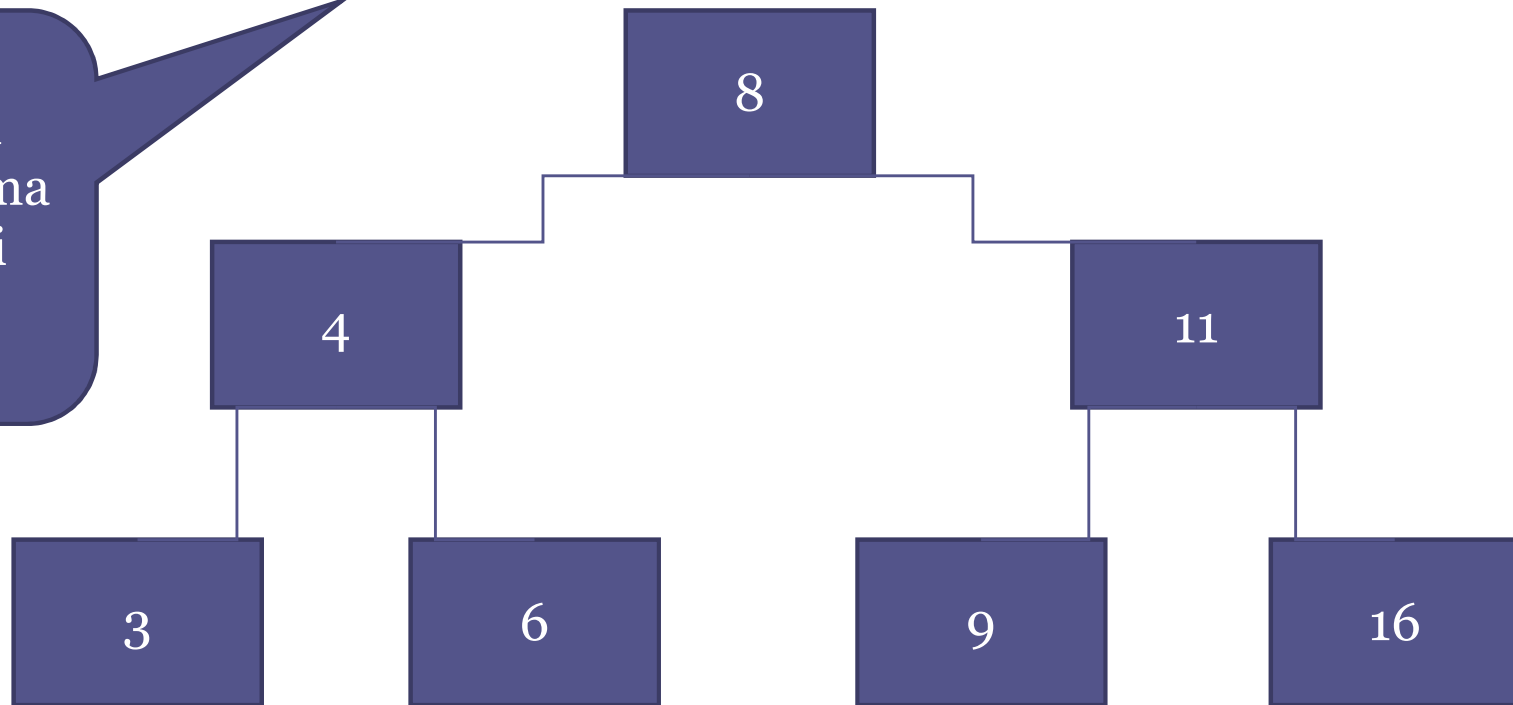


Arbori binari

- Pentru reprezentarea în Prolog, presupunem că fiecare nod are câte două legături către alți arbori:
 - una către subarborele stâng
 - una către subarborele drept

$\text{arb}(8, \text{arb}(4, \text{arb}(3, n, n), \text{arb}(6, n, n)), \text{arb}(11, \text{arb}(9, n, n), \text{arb}(16, n, n)))$

Se reprezinta
asadar sub forma
unei structuri
compuse.



Arbori binari

- Scrieți un predicat Prolog care să calculeze suma elementelor arborelui.

suma(n, 0).

suma(arb(R, n, n), R).

suma(arb(Radacina, S, D), Suma) :- suma(S, S1),
 suma(D, S2), Suma is Radacina + S1 + S2.

1 ?- suma(arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, n), arb(16, n, n))), S).

S = 57 |

Arbori binari

- Scrieți un predicat Prolog care să verifice existența unui număr dat într-un arbore binar.

```
cauta(n, _):- write('Nu exista.').
```

```
cauta(arb(X, _, _), X):- write('Numarul exista.').
```

```
cauta(arb(Rad, S, _D), X) :- X < Rad, cauta(S, X).
```

```
cauta(arb(_Rad, _S, D), X) :- cauta(D, X).
```

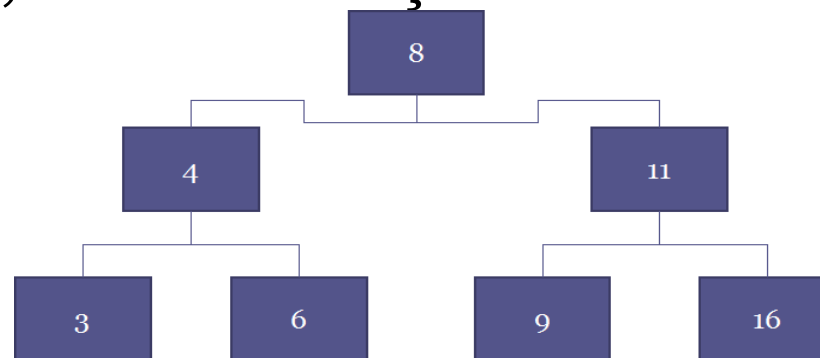
```
5 ?- cauta(arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, n), arb(16, n, n))), 9).  
Numarul exista.
```

Yes

```
6 ?- cauta(arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, n), arb(16, n, n))), 5).  
Nu exista.
```

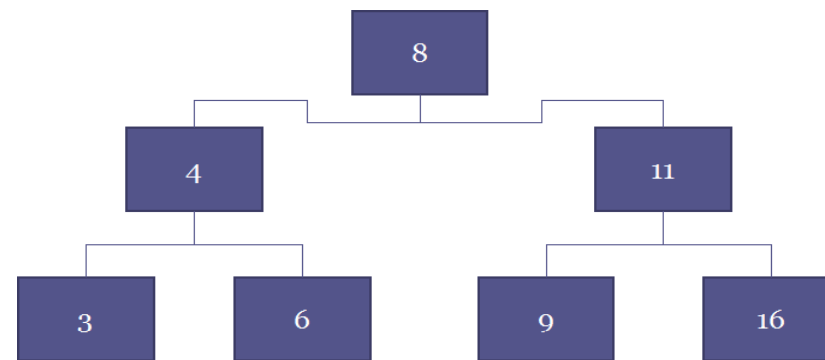
Arbori binari

- Scrieți predicate Prolog care să realizeze parcurgerea unui arbore binar în preordine, în inordine și în postordine.



- Preordine:
 1. Vizitam radacina.
 2. Vizitam subarborele stang in preordine.
 3. Vizitam subarborele drept in preordine.
- Parcurgerea:
 - 8, 4, 3, 6, 11, 9, 16.

Parcurgerea in preordine



- `preord(n).`
- `preord(arb(Rad, S, D)) :-writeln(Rad),
preord(S),
preord(D).`

```
8 ?- preord(arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, n), arb(16, n, n)))).
```

```
8
```

```
4
```

```
3
```

```
6
```

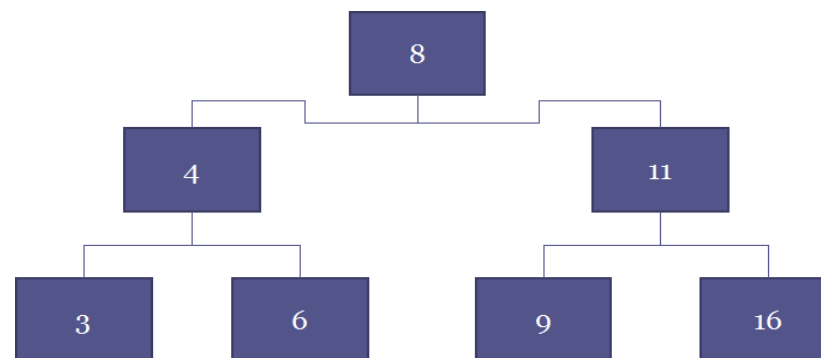
```
11
```

```
9
```

```
16
```

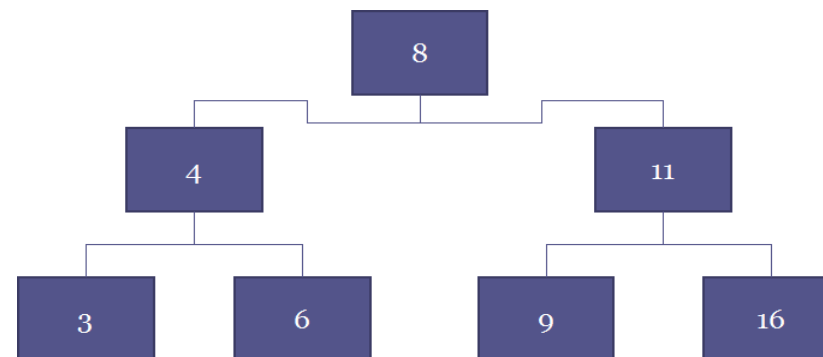
Parcurgerea in inordine

- Inordine:
 1. Vizitam subarborele stang in inordine.
 2. Vizitam radacina.
 3. Vizitam subarborele drept in inordine.
- Parcurgerea:
 - 3, 4, 6, 8, 9, 11, 16



Parcurgerea in postordine

- Postordine:
 1. Vizitam subarborele stang in postordine.
 2. Vizitam subarborele drept in postordine.
 3. Vizitam radacina.
- Parcurgerea:
 - 3, 6, 4, 9, 16, 11, 8



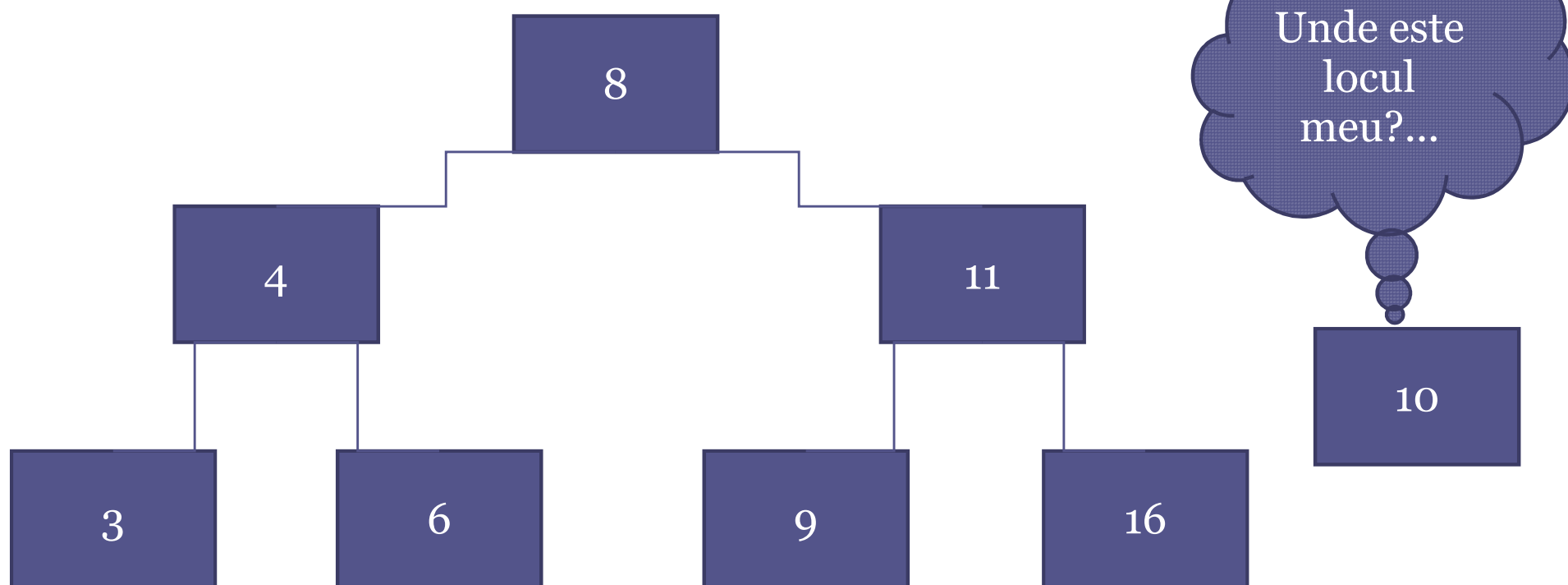
Inordine si postordine?

- Cum se implementeaza in Prolog?
 - Solutia este triviala si ramane ca tema.



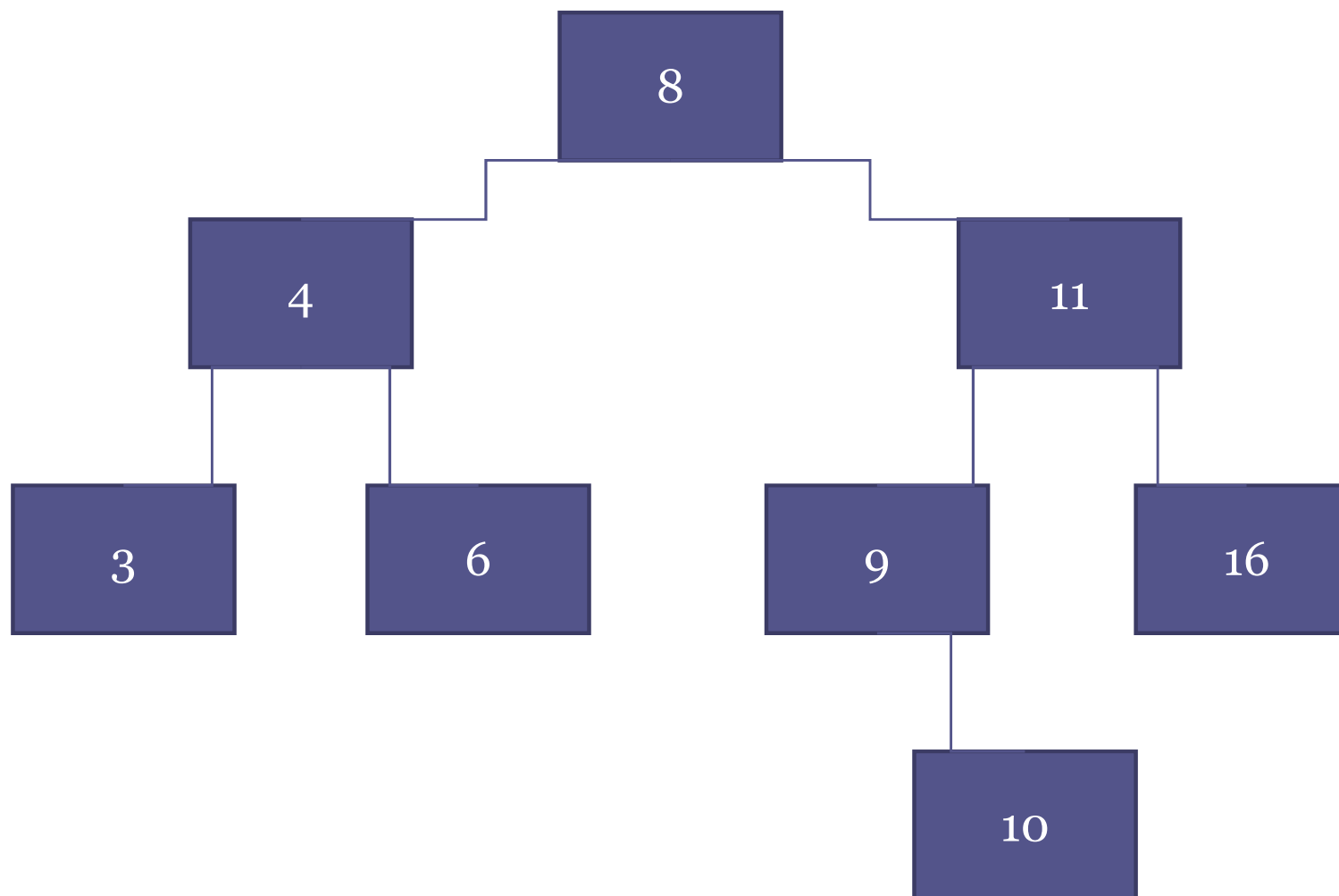
Inserarea unui element intr-un arbore binar

- Realizați un predicat Prolog care să realizeze inserarea unui element in cadrul arborelui.



Inserarea unui element intr-un arbore binar

- Realizați un predicat Prolog care să realizeze inserarea unui element in cadrul arborelui.



Inserarea unui element intr-un arbore binar

`ins(Val, n, arb(Val, n, n)).`

`ins(Val, arb(Rad, L_T, R_T), Rez) :- Val < Rad,
 ins(Val, L_T, Rez1), Rez = arb(Rad, Rez1, R_T).`

`ins(Val, arb(Rad, L_T, R_T), Rez) :- Val > Rad,
 ins(Val, R_T, Rez1), Rez = arb(Rad, L_T, Rez1).`

2 ?- `ins(10, arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, n), arb(16, n, n))), Rez).`

`Rez = arb(8, arb(4, arb(3, n, n), arb(6, n, n)), arb(11, arb(9, n, arb(10, n, n)), arb(16, n, n)))`



Intrari si iesiri in Prolog

Altfel spus, cum citim dintr-un fisier si cum scriem intr-unul folosind Prologul.

Intrari si iesiri in Prolog



- Orice sursă sau destinație de date este numită în Prolog *stream* (canal de intrare, sau de ieșire).
- Cele mai utilizate predicate pentru citirea și scrierea datelor sunt, evident, *read* și *write*.

Vom folosi si altele cand vom invata sa lucram cu caractere si string-uri in Prolog.



Intrari si iesiri in Prolog

- Atat `write/1` cat si `read/1` au un singur argument și folosesc canalul *curent* de ieșire, respectiv de intrare.
 - Cele predefinite sunt ecranul și tastatura.
 - ? – `read(X)`, `write('Am citit termenul')`, `tab(1)`, `write(X)`.
 - La citire, dupa scrierea termenului care trebuie citit, se pune punct (.).

Intrari in Prolog



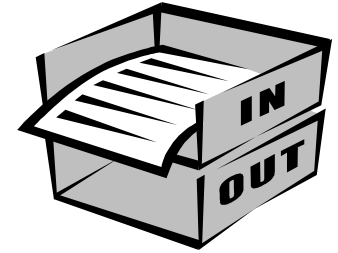
- Pentru a citi dintr-un fișier nu trebuie decât să facem din fișierul respectiv canalul curent.
- Predicatele care fac acest lucru sunt:
 - **see(F)** – fișierul dat ca argument devine fișier de intrare curent.
 - Fișierul F este deschis pentru citire, iar pointerul de fișier este poziționat la începutul lui.
 - **seen** – închide fișierul de intrare curent și stabilește canalul de intrare tastatura.



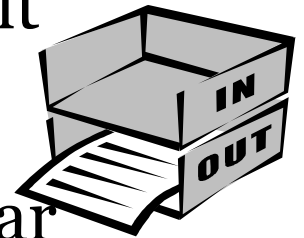
Iesiri in Prolog

- Pentru a scrie într-un fișier trebuie să facem din fișierul respectiv canalul curent.
- Predicatele care fac acest lucru sunt:
 - **tell(F)** – deschide fișierul F pentru scriere și îl face fișier de ieșire curent.
 - Dacă fișierul există deja, conținutul său este șters, altfel, fișierul este creat.
 - **append(F)** – face același lucru ca și `tell/1`, cu deosebirea că, dacă fișierul există, conținutul său nu este șters, ci se scrie în continuare.
 - **told** – închide fișierul de ieșire curent stabilind ca stream de ieșire ecranul.

Intrari/iesiri in Prolog



- Predicatul *read(-Term)* citește în variabila *Term* următorul termen din fișierul de intrare.
 - Termenul citit trebuie să se termine cu caracterul punct în cadrul fișierului.
- Există o constantă specială în Prolog, *end_of_file*, care este returnată atunci când s-au citit toate datele dintr-un fișier.
- Un fișier Prolog poate fi încărcat din interiorul unui alt fișier, cu ajutorul predicatului *consult(NumeFisier)*.
- *nl* provoacă trecerea la o linie următoare (*new line*), iar *tab(N)* adaugă *N* spații față de poziția la care este situat pointerul în canalul de ieșire curent.



Exemplu

- Avem fișierul de intrare *in.txt* care conține câte un număr urmat de caracterul punct pe fiecare linie. Scrieți în fișierul *pare.txt* numerele conținute în *in.txt* care sunt pare, iar în *impare.txt* numerele care sunt impare.

Separarea elementelor pare de cele impare

`separ([], [], []).`

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

`separ([P|R1], [P|R2], L2) :- separ(R1, R2, L2).`

1 ?- `separ([2, 7, 11, 14], X, Y).`

`X = [2, 14]`
`Y = [7, 11]` |

Afisarea elementelor unei liste

```
afis([]).
```

```
afis([P|R]) :- write(P), nl, afis(R).
```

```
1 ?- afis([2, 7, 11, 14]).
```

```
2
```

```
7
```

```
11
```

```
14
```

Citirea din *in.txt* si scrierea in *pare.txt* si *impare.txt*

exemplu :- see('in.txt'), citesc([]), seen, write('Totul este gata').

```
citesc(L) :- read(N), N \= end_of_file, append(L, [N], Rez),  
            citesc(Rez).
```

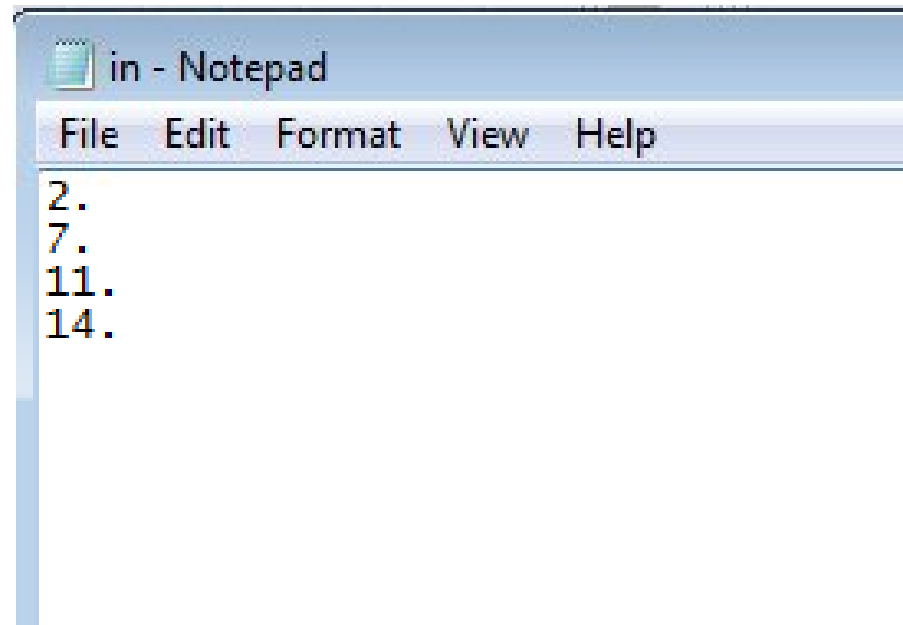
```
citesc(L) :- separ(L, Pare, Impare), pare(Pare), impare(Impare).
```

```
pare(L) :- tell('pare.txt'), afis(L), told.
```

```
impare(L) :- tell('impare.txt'), afis(L), told.
```

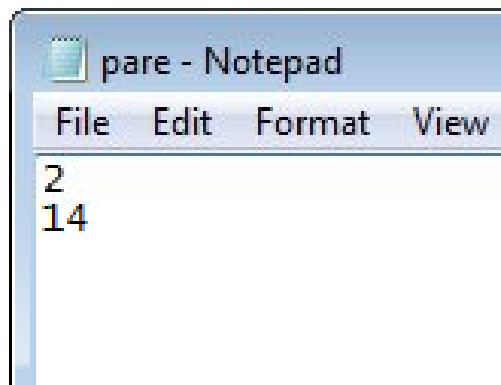
Rularea programului

- Scriem mai intai intr-un fisier text cateva numere urmate de caracterul punct, fiecare aflate pe cate un rand.
 - Il salvam la aceeasi locatie unde se afla si programul.

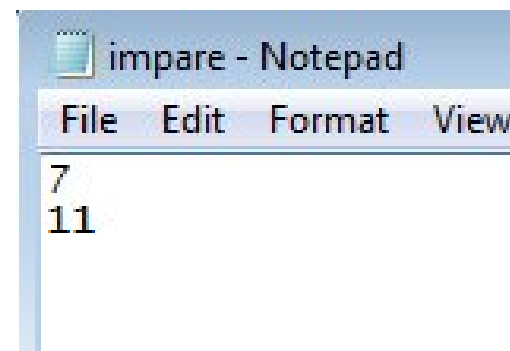


Rularea programului

1 ?- exemplu.
Total este gata



```
pare - Notepad
File Edit Format View
2
14
```



```
impare - Notepad
File Edit Format View
7
11
```

Alt exemplu

- Dacă fișierul *in.txt* conține câte un număr urmat de punct pe fiecare linie, scrieți un predicat Prolog care să introducă în fișierul *suma.txt* mesajul *Suma este* urmat de valoarea sumei numerelor din *in.txt*.
- Cum se rezolva?
 - Avem același cod de la programul precedent pentru citirea elementelor într-o listă.
 - Se calculează apoi suma elementelor din listă și se scrie aceasta în fișierul *suma.txt*.

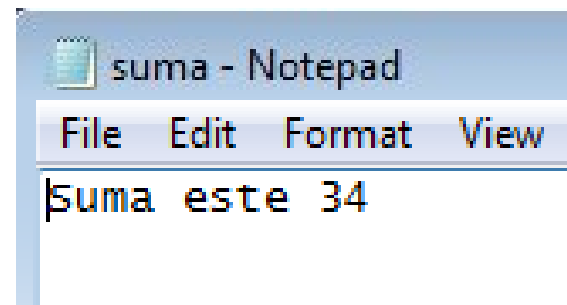
Sau... gasim o alta rezolvare

```
suma:- see('in.txt'), tell('suma.txt'), calc(0), told, seen.
```

```
calc(S) :- read(N), N \= end_of_file, S1 is S + N, calc(S1).
```

```
calc(S) :- write('Suma este '), write(S).
```

```
1 ?- suma.
```



Selectati numerele prime

- Având fișierul *in.txt* dat ca in exemplele precedente, realizați un predicat Prolog care să scrie în fișierul *prime.txt* numai acele numere care sunt prime.
 - Citim toate numerele intr-o lista, ca la exemplul initial (predicatul *citesc/1*), apoi selectam numerele prime intr-o lista pe care o *afisam* apoi in fisierul tinta.
 - Simplu, nu?



Verificarea daca un numar este prim

$\text{prim}(N) :- \text{prim}(N, 2).$

$\text{prim}(N, K) :- I \text{ is } N/2, K > I.$

$\text{prim}(N, K) :- R \text{ is } N \bmod K, R \neq 0, K1 \text{ is } K + 1,$
 $\text{prim}(N, K1).$

1 ?- prim(17).

Yes

2 ?- prim(18).

No

3 ?- |

Fara sa folosim o lista temporara...

```
prime :- see('in.txt'), tell('prime.txt'), told, calcul, seen.
```

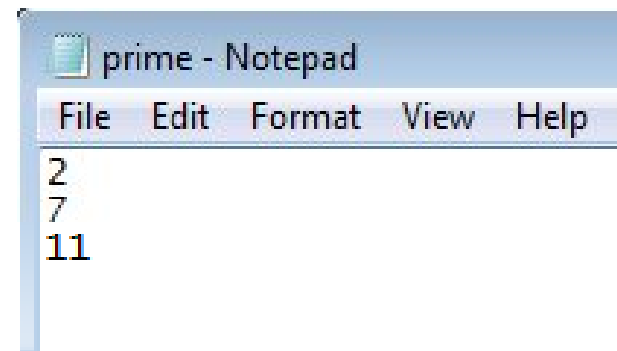
```
calcul :- read(X), X \= end_of_file, verific(X), calcul.  
calcul.
```

```
verific(X) :- prim(X), append('prime.txt'), write_ln(X),  
             told.
```

```
verific(_).
```

Selectati numerele prime

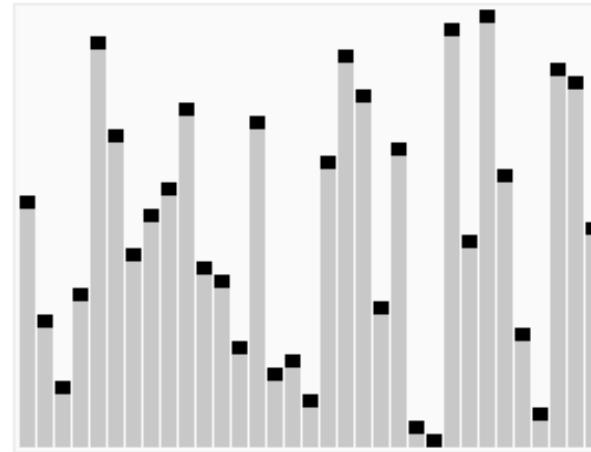
1 ?- prime.



Ordonarea unui sir de numere

- Având în fișierul de intrare *in.txt* câte un număr urmat de caracterul punct pe fiecare linie, construiți un predicat Prolog care să scrie în fișierul *ordonat.txt* șirul de numere ordonat crescător.
 - Citim numerele din *in.txt* într-o lista, le ordonăm în alta lista și le scriem apoi în fișierul *ordonat.txt*.
 - O să facem în continuare numai ordonarea elementelor unei liste.
 - Pentru aceasta, vom folosi metoda *quicksort* care utilizează mecanismul *divide et impera*.

Ordonarea elementelor unei liste



```
sortez([], []).
```

```
sortez([P|Rest], Lrez):- selectez(P, Rest, Mici, Mari),
    sortez(Mici, MiciSort), sortez(Mari, MariSort),
    append(MiciSort, [P|MariSort], Lrez).
```

```
selectez(_, [], [], []).
```

```
selectez(P, [P1|Rest], [P1|Mici], Mari):- P1 < P,
    selectez(P, Rest, Mici, Mari).
```

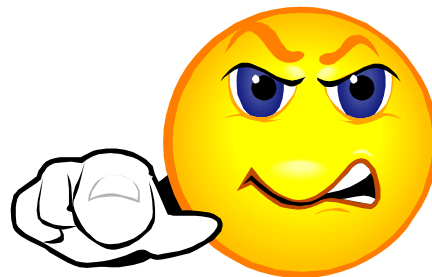
```
selectez(P, [P1|Rest], Mici, [P1|Mari]):- selectez(P, Rest,
    Mici, Mari).
```

Rularea algoritmului de sortare

```
1 ?- sortez([4, -12, 41, 19, 2], L).
```

```
L = [-12, 2, 4, 19, 41] |
```

- Citirea elementelor dintr-un fisier si scrierea elementelor sortate in fisierul sortat urmeaza sa fie facute de...



Pe saptamana viitoare!

