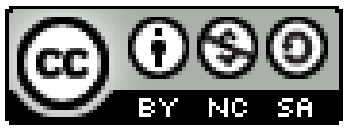
ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Υπολογιστική Λογική και Λογικός Προγραμματισμός

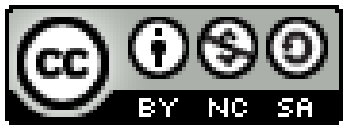
Ενότητα 8: Γλώσσα Prolog: Διαχείριση δεδομένων, προγράμματος και λύσεων στην Prolog

Νίκος Βασιλειάδης, Αναπλ. Καθηγητής
Τμήμα Πληροφορικής



Άδειες Χρήσης

- Το παρόν εκπαιδευτικό υλικό υπόκειται σε άδειες χρήσης Creative Commons.
- Για εκπαιδευτικό υλικό, όπως εικόνες, που υπόκειται σε άλλου τύπου άδειας χρήσης, η άδεια χρήσης αναφέρεται ρητώς.



Χρηματοδότηση

- Το παρόν εκπαιδευτικό υλικό έχει αναπτυχθεί στα πλαίσια του εκπαιδευτικού έργου του διδάσκοντα.
- Το έργο «Ανοικτά Ακαδημαϊκά Μαθήματα στο Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης» έχει χρηματοδοτήσει μόνο την αναδιαμόρφωση του εκπαιδευτικού υλικού.
- Το έργο υλοποιείται στο πλαίσιο του Επιχειρησιακού Προγράμματος «Εκπαίδευση και Δια Βίου Μάθηση» και συγχρηματοδοτείται από την Ευρωπαϊκή Ένωση (Ευρωπαϊκό Κοινωνικό Ταμείο) και από εθνικούς πόρους.

ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Γλώσσα Prolog: Διαχείριση δεδομένων, προγράμματος και λύσεων στην Prolog

Έλεγχος Τύπου Δεδομένων (1/4)

- Οι μεταβλητές στην Prolog δεν έχουν τύπο και άρα μπορούν να ενοποιηθούν με οποιονδήποτε όρο.
- Σε αρκετές περιπτώσεις είναι απαραίτητο να μπορούμε να καθορίσουμε το τύπο μιας μεταβλητής.
- Η Prolog παρέχει ένα σύνολο κατηγορημάτων για το σκοπό αυτό.
- **var(X)**: Το κατηγόρημα πετυχαίνει όταν η **X** είναι ελεύθερη μεταβλητή (δεν έχει πάρει ακόμη τιμή).

?- var(Date).

yes

?- var(32) .

no

?- X=5, var(X) .

no

Έλεγχος Τύπου Δεδομένων (2/4)

- **nonvar(X)**: Το κατηγόρημα πετυχαίνει όταν το **X** είναι σταθερά ή μεταβλητή η οποία έχει πάρει κάποια τιμή.

?- nonvar(date).

yes

?- nonvar(X).

no

?- X=5, nonvar(X).

yes

- **atom(X)**: Το κατηγόρημα πετυχαίνει όταν το **X** είναι άτομο.

?- atom(john) .

yes

?- atom(X).

no

?- atom(32).

no

?- atom(f(2,3)).

no

Έλεγχος Τύπου Δεδομένων (3/4)

- **integer(X)**: Το κατηγόρημα πετυχαίνει όταν το **X** είναι ακέραιος αριθμός.

?- integer(1298) .

yes

?- integer(32.9).

no

- **float(X)**: Πετυχαίνει όταν το **X** είναι αριθμός με υποδιαστολή.

?- float(1298) .

no

?- float(32.9).

yes

Έλεγχος Τύπου Δεδομένων (4/4)

- **atomic(X)**: Πετυχαίνει όταν το **X** είναι αριθμός ή άτομο.

?- atomic(prolog).

yes

?- atomic(f(4,5)).

no

?- atomic(X).

no

?- atomic(4).

yes

- **compound(X):** Πετυχαίνει όταν το **X** είναι σύνθετος όρος.

?- compound(f(4,5)).

yes

Έλεγχος Τύπου: Παράδειγμα (1/2)

- Να οριστεί το κατηγόρημα **print_type,** το οποίο να ζητάει από το χρήστη να πληκτρολογήσει έναν όρο και να τυπώνει στην οθόνη τον τύπο του όρου (π.χ. ακέραιος, άτομο, κλπ).

?- print_type.

Give a term: *(έστω ότι πληκτρολογούμε :)* ***5.***

Term 5 is of type: integer

yes

?- print_type.

Give a term: ***a.***

Term a is of type: atom

yes

?- print_type.

Give a term: ***A.***

Term _312234 is of type: variable

yes

Έλεγχος Τύπου: Παράδειγμα (2/2)

```
print_type :-
    write('Give a term: '),    read(X),
    find_type(X,Type),
    write('Term '), write(X),
    write(' is of type: '), write(Type), nl.

find_type(X,atom) :- atom(X).
find_type(X,integer) :- integer(X).
find_type(X,float) :- float(X).
find_type(X,compound) :- compound(X).
find_type(X,variable) :- var(X).
```

Σύνθεση/Διάσπαση Σύνθετων Όρων

- Η Prolog παρέχει τη δυνατότητα σύνθεσης και διάσπασης σύνθετων όρων.
- *Γιατί?*
  - Οι σύνθετοι όροι μπορούν να μετατραπούν σε λίστες και στη συνέχεια να τους χειριστούμε με αναδρομικές διαδικασίες.
    - Π.χ. να ψάξουμε να βρούμε αν ένας όρος έχει μέσα του ένα σύμβολο, χωρίς να ξέρουμε ακριβώς σε ποια θέση.

      student(name(nick),aem(1453),year(3))
  - Μπορώ να «κατασκευάσω» μέσω προγραμματισμού έναν σύνθετο όρο για διάφορες χρήσεις.
    - Π.χ. ο σύνθετος όρος αντιπροσωπεύει κλήση (δυναμική-μεταβλητή κλήση): **member(X,[a,b,c,d]).**

Κατηγόρημα =../2

- Το κατηγόρημα "**=..**" χρησιμοποιείται και για σύνθεση και για διάσπαση σύνθετων όρων.
  - Ονομάζεται ***univ*** (προφέρεται ***γιουνίβ***).
- Το κατηγόρημα πετυχαίνει όταν:
  - το πρώτο όρισμα του είναι ένας σύνθετος όρος.
  - ενώ το δεύτερο μια λίστα,
  - Το πρώτο στοιχείο της λίστας είναι το συναρτησιακό σύμβολο του σύνθετου όρου ενώ τα υπόλοιπα στοιχεία της είναι τα ορίσματα του.

Term =.. [Functor | Arguments]

Σύνθεση/Διάσπαση Σύνθετων Όρων

- Η ακόλουθη κλήση κάνει αποσύνθεση ενός σύνθετου όρου σε λίστα:

?- parent(john,mary) =.. List.

List = [parent,john,mary]

?- parent(john,mary)=..[parent,john,mary].

yes

- Αν η μεταβλητή υπάρχει αριστερά ενώ δεξιά υπάρχει λίστα, τότε το κατηγόρημα univ κάνει σύνθεση όρου:

?- P =.. [point,3,2]. **?- point(X,Y) =.. [point,3,2].**

P = point(3,2) **X=3, Y=2**

Κατηγόρημα functor/3 (1/2)

- Μπορεί να χρησιμοποιηθεί για την αποσύνθεση όρων

functor(Term, Functor, No_of_arguments)

  - **Term** είναι κάποιος σύνθετος όρος.
  - **Functor** είναι το συναρτησιακό σύμβολο του.
  - **No_of_arguments** είναι ο αριθμός των ορισμάτων του.

- Χρησιμοποιείται κυρίως για να πάρουμε το συναρτησιακό σύμβολο ενός σύνθετου όρου.

?- functor(student(name(john),id(324)),Fun,Args).

Fun = student

Args = 2

Κατηγόρημα functor/3 (2/2)

- Μπορεί να χρησιμοποιηθεί για τη δημιουργία όρων των οποίων τα ορίσματα είναι μεταβλητές χωρίς τιμή.
  - Αυτή η χρήση δεν συνηθίζεται πολύ.

?- functor(Term,driver,2).

Term = driver(Z, B)

Κατηγόρημα arg/3 (1/2)

- Επιστρέφει το ν-οστό όρισμα ενός σύνθετου όρου.

arg(N,Term,Argument)

  - **N** η θέση του ορίσματος που θέλουμε να πάρουμε από τον σύνθετο όρο **Term**.
  - Το όρισμα ταυτοποιείται ("επιστρέφεται") με την μεταβλητή **Argument**.

?- arg(2,student(name(john),class(b)),class(b)).

yes

?- arg(1,think(positive),X).

X = positive

Κατηγόρημα arg/3 (2/2)

- Το κατηγόρημα **arg/3** μπορεί να φανεί ιδιαίτερα χρήσιμο όταν θέλουμε να δώσουμε τιμή σε κάποιο όρισμα ενός σύνθετου όρου χωρίς να προχωρήσουμε σε αποσύνθεση και επανασύνθεση του όρου.

?- arg(1,student(X,aem(1234)),name(john)).

X = name(john).

Ισοδυναμία univ/functor/arg

- Το univ μόνο του ή το functor/arg σε συνδυασμό μπορούν να αντικαταστήσουν το ένα το άλλο.
- Δεν χρειάζονται δηλαδή όλα αυτά τα κατηγορήματα, αλλά αν υπάρχει το ένα μπορεί να χρησιμοποιηθεί για να κατασκευαστεί το άλλο.

Κατασκευή functor/arg από univ (1/2)

```
my_functor(Term,Functor,Arity) :-
        Term =.. List,
        List = [Functor|Arguments],
        length(Arguments,Arity).

my_arg(N,Term,Arg) :-
        Term =.. List,
        List = [Functor|Arguments],
        nth_member(N,Arg,Arguments).
```

Κατασκευή functor/arg από univ (2/2)

```
my_functor(Term,Functor,Arity) :-
        Term =.. List,
        List = [Functor|Arguments],
        length(Arguments,Arity).

my_arg(N,Term,Arg) :-
        Term =.. List,
        List = [Functor|Arguments],
        nth1(N,Arguments,Arg).
```

Υπάρχει έτοιμο στην SWI-Prolog

Κατασκευή univ από functor/arg

```
univ(Term,List) :-
        nonvar(Term),
        functor(Term,Functor,Arity),
        univ_aux(Term,1,Arity,Arguments),
        List = [Functor| Arguments].
univ(Term,List) :-
        var(Term),
        List = [Functor| Arguments],
        length(Arguments,Arity),
        functor(Term,Functor,Arity),
        univ_aux(Term,1,Arity,Arguments).
```

Ανάλυση του Term σε Functor και Arity

Σύνθεση του Term από Functor και Arity

Κατασκευή univ από functor/arg

Βοηθητικό κατηγόρημα

```
univ_aux(_,N,Arity,[]) :-
        N > Arity.
univ_aux(Term,N,Arity,[Arg|Rest]) :-
        N =< Arity,
        arg(N,Term,Arg),
        N1 is N + 1,
        univ_aux(Term,N1,Arity,Rest).
```

Ταυτοποίηση N-οστού Arg του Term με το N-οστό στοιχείο της λίστας

Δουλεύει και προς τις 2 κατευθύνσεις

Έλεγχος Τύπου & Σύνθεση/Διάσπαση Όρων: Παράδειγμα (1/2)

- Το κατηγόρημα ελέγχει αν ένας σύνθετος όρος περιέχει ελεύθερες μεταβλητές.
  - Σε περίπτωση που έχει, το κατηγόρημα αποτυγχάνει.

```
?- ground_term(a).
yes
?- ground_term(A).
no
```

```
?- ground_term(f(1,g(2,a))).
yes
?- ground_term(f(1,g(2,X))).
no
```

Έλεγχος Τύπου & Σύνθεση/Διάσπαση Όρων: Παράδειγμα (2/2)

- Ο όρος αποσυντίθεται χρησιμοποιώντας το κατηγόρημα "=.."
  - Ελέγχεται αν δεν είναι μεταβλητή κάθε μέλος της λίστας των ορισμάτων του σύνθετου όρου που προκύπτει από την αποσύνθεση.
  - Απαιτείται ύπαρξη βοηθητικού κατηγορήματος (**ground_aux/l**) το οποίο αναδρομικά ελέγχει όλα τα στοιχεία της λίστας.

```
ground_term(Term) :-
    atomic(Term).
ground_term(Term):-
    nonvar(Term),
    Term =.. [Functor|Arguments],
    ground_aux(Arguments).
```

```
ground_aux([]).
ground_aux([First|Rest]):-
    ground_term(First),
    ground_aux(Rest).
```

Έλεγχος Τύπου & Σύνθεση/Διάσπαση Όρων: Παράδειγμα 2 (1/2)

- Να ορισθεί η διαδικασία **subs(ST,T,ST1,T1)**, η οποία αληθεύει αν ο όρος **T1** προκύπτει από τον όρο **T** με αντικατάσταση του υπο–όρου **ST** με **ST1**.

?- subs(a+b, f(a+b), new, A).

A = f(new)

```
subs(T,T,T1,T1).          % Αντικατάστησε όλο τον όρο T=T1.
subs(_,T,_,T) :-          % Αν ο T δεν είναι σύνθετος όρος,
    atomic(T).            % μην αντικαθιστάς τίποτα.
subs(Sub,Term,Sub1,Term1) :-
    Term =..[F|Args],                 % πάρε τα ορίσματα.
    sublist(Sub,Args,Sub1,Args1),     % αντικατέστησε σε αυτά.
    Term1 =.. [F|Args1].              % Σύνθεση.
```

Έλεγχος Τύπου & Σύνθεση/Διάσπαση Όρων: Παράδειγμα 2 (2/2)

```
sublist(_,[],_,[]).
sublist(Sub,[Term|Terms],Sub1,[Term1|Terms1]): -
    % Κάθε ένας υπο-όρος αντικαθίσταται.
    subs(Sub,Term,Sub1,Term1),
    sublist(Sub,Terms,Sub1,Terms1).
```

Μεταβλητή Κλήση

- Η μεταβλητή κλήση μας δίνει τη δυνατότητα να τοποθετήσουμε μια μεταβλητή στη θέση μιας κλήσης στο σώμα ενός κανόνα ή ερώτησης.
  - Η μεταβλητή αυτή παίρνει τιμή και εκτελείται κατά τη στιγμή της εκτέλεσης (at run-time).
- Π.χ. έστω το ακόλουθο πρόγραμμα:

```
b(3).
a(X) :- X.
```

- και η ερώτηση:

```
?- a(b(3)).
yes
```

Η αρχική ερώτηση ανάγεται λόγω της μεταβλητής κλήσης στην ερώτηση ***?- b(3).****, της οποίας το αποτέλεσμα θα αποτελέσει και αποτέλεσμα της αρχικής ερώτησης.*

Σε πολλές εκδόσεις της Prolog υποστηρίζεται το κατηγόρημα ***call(X)*** *που εκτελεί την ίδια λειτουργία.*

```
a(X) :- call(X).
```

Σύγκριση Σύνθετων Ορών και Ατομικών Τύπων

- Η μεταβλητή κλήση είναι δυνατή λόγω της ομοιότητας που παρουσιάζουν οι *σύνθετοι όροι* και οι *ατομικοί τύποι.*
  - Αφορά **ΜΟΝΟ** στη σύνταξη.
- Η διαφορά τους έγκειται στη θέση τους μέσα στη πρόταση, καθώς και στην ερμηνεία τους:
  - Ο ατομικός τύπος παριστάνει μια σχέση μεταξύ των όρων του, που ισχύει ή δεν ισχύει (δηλαδή μπορεί να αποδοθεί τιμή true ή false).

father(nick). *% Ο Nick είναι πατέρας.*

  - Ο σύνθετος όρος αναπαριστά μια συνάρτηση των όρων του, χωρίς τιμή αλήθειας.

Y = father(nick) *% Ο Y είναι ο πατέρας του Nick.*

Σύνθεση Ορών και Μεταβλητή Κλήση

- Η μεταβλητή κλήση συνδυάζεται με το univ για σύνθεση όρων.
  - Αρχικά κατασκευάζεται δυναμικά μια κλήση με τη βοήθεια του univ και μετά καλείται με μεταβλητή κλήση.

?- X =.. [member,a,[a,b,c]], call(X).

X = member(a,[a,b,c])

ή

?- X =.. [member,a,[a,b,c]], X.

Παράδειγμα μεταβλητής κλήσης (1/2)

- Ποιο είναι το αποτέλεσμα του προγράμματος, αν δώσουμε σαν είσοδο: **a(K).**

```
a(5).

?- read(X), X.

|: a(K).

X=a(5)
```

Παράδειγμα μεταβλητής κλήσης (2/2)

- Να γραφεί πρόγραμμα το οποίο να διαβάζει εντολές από το πληκτρολόγιο (add, sub,…) και να τις εκτελεί (διερμηνέας).
- Το πρόγραμμα θα σταματά όταν θα διαβάσει μια εντολή που δεν υπάρχει.

```
add(X,Y) :- Z is X+Y, write('Result = '), write(Z), nl.

sub(X,Y) :- Z is X-Y, write('Result = '), write(Z), nl.

div(X,Y) :- Z is X/Y, write('Result = '), write(Z), nl.

mul(X,Y) :- Z is X*Y, write('Result = '), write(Z), nl.

run :- write('Entoli = '), read(E), E, nl, run.

run.
```

?-run.

Entoli = *(έστω ότι πληκτρολογούμε :)* **add(5,7).**

Result = 12

Εναλλακτική Λύση

```
add(X,Y,Z) :- Z is X+Y.          sub(X,Y,Z) :- Z is X-Y.
div(X,Y,Z) :- Z is X/Y.          mul(X,Y,Z) :- Z is X*Y.
run :-
    write('Entoli = '), read(E),
    E =.. [Op,X,Y], E1 =.. [Op,X,Y,Z],
    call(E1),
    write('Result = '), write(Z), nl, run.
run.
```

?-run.

Entoli = add(5,7).

Result = 12

add(5,7) →
[add,5,7] →
[add,5,7,Z] →
add(5,7,Z)

Προηγούμενο παράδειγμα χωρίς μεταβλητή κλήση

```
run :-
    write(‘Entoli = ’), read(E),
    write('First number:'), read(X),
    write('Second number:'), read(Y),
    operation(E,X,Y,Z),
    write(‘Result = ’), write(Z), nl, run.
run.
operation(add,X,Y,Z) :- Z is X+Y.
operation(sub,X,Y,Z) :- Z is X-Y.
operation(div,X,Y,Z) :- Z is X/Y.
operation(mul,X,Y,Z) :- Z is X*Y.
?-run.
Entoli = add.
First number: 5.
Second number: 7.
Result = 12
```

Παράδειγμα Μεταβλητής Κλήσης

- Έστω ότι έχουμε καταχωρημένο όλο το προσωπικό μιας εταιρείας σε γεγονότα της μορφής:

secretary(name(mary), id(43)).

manager(name(john),id(23)).

manager(name(nick),id(55)).

programmer(name(daffy),id(33)).

- Θέλουμε ένα κατηγόρημα, το οποίο να επιστρέφει το ID του εργαζομένου, όταν δίνουμε την θέση του και το όνομα του:

```
find_person(Name, Position,ID):-
        Q =.. [Position,name(Name),id(ID)],
        Q.              % ή call(Q)
```

Ερωτήσεις προς το πρόγραμμα

?- find_person(john,manager,ID).

ID = 23.

?- find_person(Name,manager,33).

Name = daffy.

?- find_person(Name,manager,ID).

Name = john, ID = 23 ;

Name = nick, ID = 55

?- find_person(john,Position,23).

INSTANTIATION_ERROR

- Στην τελευταία περίπτωση το σύστημα επέστρεψε λάθος καθώς το συναρτησιακό σύμβολο του σύνθετου όρου της μεταβλητής κλήσης (Position) ήταν ελεύθερη μεταβλητή.

Μετα-προγραμματισμός

- Η δυνατότητα δυναμικής δημιουργίας σύνθετων όρων και δυναμικής κλήσης τους (λόγω της ισοδυναμίας της σύνταξης ατομικών προτάσεων και σύνθετων όρων), αποτελεί πολύ προηγμένο εργαλείο στα χέρια του προγραμματιστή.
  - Ονομάζεται ***μετα-προγραμματισμός***
- Μπορούμε να ορίσουμε νέες γλώσσες προγραμματισμού με την χρήση της Prolog και να φτιάξουμε και τον διερμηνέα τους ή τον μεταφραστή τους.
  - Μπορούμε να την αλλάξουμε ακόμα και την Prolog.

Προσθαφαίρεση προτάσεων

- Τα γεγονότα και οι κανόνες ενός προγράμματος Prolog αποθηκεύονται στη μνήμη του συστήματος.
  - Συνήθως «φορτώνονται» από κάποιο αρχείο κειμένου.
- Υπάρχει η δυνατότητα τροποποίησης των περιεχομένων της μνήμης της Prolog (δηλ. του προγράμματος), κατά την διάρκεια εκτέλεσης του.
- Η προσθήκη/αφαίρεση γίνεται μέσω 3 κατηγορημάτων:
  - **asserta(X)**: Προσθέτει στην μνήμη το **X** (γεγονός ή κανόνας) τοποθετώντας το πριν από τις ήδη υπάρχουσες προτάσεις του ίδιου κατηγορήματος.
  - **assertz(X)**: Προσθέτει το **X** μετά από τις προτάσεις του ίδιου κατηγορήματος.
  - **retract(X)**: Αφαιρεί από την μνήμη το γεγονός ή τον κανόνα **X**.

Παράδειγμα (1/3)

- Έστω ότι στη μνήμη της Prolog υπάρχει το γεγονός **father(nick,mary).** και κάνουμε την ερώτηση:

?- father(nick,X).

X=mary

- Αν προστεθεί ένα γεγονός πάνω από τα υπόλοιπα:

?- asserta(father(nick,john)).

yes

?- father(nick,X).

X = john;

X = mary

Παράδειγμα (2/3)

- Αν προστεθεί ένα γεγονός κάτω από τα υπόλοιπα:

?- assertz(father(nick,anna)).

yes

?- father(nick,X).

X=john;

X=mary;

X=anna

Παράδειγμα (3/3)

- Αν αφαιρεθεί ένα γεγονός:

?- retract(father(Y,mary)).

Y=nick

?- father(nick,X).

X=john;

X=anna

Παρατηρήσεις

- Η εκτεταμένη χρήση των κατηγορημάτων **assert/retract** μπορεί να οδηγήσει σε προγράμματα τα οποία είναι πολύ δύσκολο να αποσφαλματωθούν.
  - Για το λόγο αυτό θα πρέπει η χρήση τους να περιορίζεται μόνο στις απόλυτα αναγκαίες περιπτώσεις.
- Μερικές υλοποιήσεις της Prolog επιτρέπουν μόνο την εισαγωγή γεγονότων και όχι κανόνων.
- Σε μερικές υλοποιήσεις της Prolog, απαιτούνται δηλώσεις.

:- dynamic functor/arity.

  - Δηλώνουν ποια κατηγορήματα μπορούν να τροποποιηθούν δυναμικά μέσω των **assert** και **retract**.

Χρήσεις assert/retract

- Προσθήκη στο πρόγραμμα της λύσης ενός προβλήματος:

χωρίς τιμή

?- solve(Problem, Answer),

asserta(solve(Problem, Answer)).

με τιμή

- Αν υποβληθεί στο πρόγραμμα το ίδιο "**Problem**" δεν θα χρειαστεί να ξανα-υπολογιστεί το "**Answer**", γιατί θα υπάρχει σαν γεγονός (fact).
  - Η τεχνική ονομάζεται caching ή memoing.

Παράδειγμα: Παραγοντικό

factorial1(0,1). **factorial1(N,F) :-** **N1 is N - 1,** **factorial1(N1,F1),** **F is N * F1.**	**factorial2(0,1).** **factorial2(N,F) :-** **N1 is N - 1,** **factorial2(N1,F1),** **F is N * F1,** **asserta(factorial2(N,F)).**
% Εκτελούνται 5 αναδρομές **?- factorial1(5,A).** **A = 120**	*% Εκτελούνται 5 αναδρομές* **?- factorial2(5,A).** **A = 120**
% Εκτελούνται 6 αναδρομές **?- factorial1(6,A).** **A = 720**	*% Εκτελείται 1 αναδρομή!!* **?- factorial2(6,A).** **A = 720**

Τι γίνεται στην Βάση Γνώσης; (1/2)

```
?- listing.
factorial1(0, 1).
factorial1(A, C) :-
        B is A-1,
        factorial1(B, D),
        C is A*D.
```

```
?- listing.
factorial1(0, 1).
factorial1(A, C) :-
        B is A-1,
        factorial1(B, D),
        C is A*D.
```

```
?- factorial1(5,A).
A = 120
```

Τι γίνεται στην Βάση Γνώσης; (2/2)

```
?- listing.
:- dynamic factorial2/2.
factorial2(0, 1).
factorial2(A, C) :-
    B is A-1,
    factorial2(B, D),
    C is A*D,
    asserta(factorial2(A, C)).

?- factorial2(5,A).
A = 120
```

```
?- listing.
factorial2(5, 120).
factorial2(4, 24).
factorial2(3, 6).
factorial2(2, 2).
factorial2(1, 1).
factorial2(0, 1).
factorial2(A, C) :-
    B is A-1,
    factorial2(B, D),
    C is A*D,
    asserta(factorial2(A, C)).
```

Χρήσεις assert/retract

- Προσομοίωση Καθολικών Μεταβλητών (global variables)
  - Αποθηκεύουμε την τιμή μιας μεταβλητής για να χρησιμοποιηθεί από όλες τις προτάσεις του προγράμματος.

- **ΠΑΡΑΔΕΙΓΜΑ:** Έστω τα ακόλουθα γεγονότα:

next(greece,turkey). **next(greece,albania).**

next(greece,bulgaria). **next(greece,fyrom).**

next(bulgaria,romania). **…**

- Να γραφεί κατηγόρημα που να υπολογίζει με πόσες χώρες συνορεύει μια συγκεκριμένη χώρα.

Παράδειγμα καθολικής μεταβλητής

```
count(Country,T) :-
        assert(counter(0)),fail.
count(Country,T) :-
        next(Country,X),
        retract(counter(T)), T1 is T + 1,
        assert(counter(T1)),
        fail.
count(Country,T) :-
        retract(counter(T)).
```

Αρχικοποίηση counter.

Παραγωγή εναλ-λακτικών λύσεων.

Επανάληψη μέσω οπισθοδρόμησης.

Μη-αναστρέψιμη ενέργεια.

Ανάκληση τελικής τιμής counter.

```
?- count(greece,N).
N=4
```

Διαχείριση Συνόλου Λύσεων

- Πολλές φορές είναι ιδιαίτερα χρήσιμο να επεξεργαστούμε **όλες** τις εναλλακτικές λύσεις σε μια κλήση.
  - Π.χ. υλοποίηση αλγορίθμων αναζήτησης.
  - Π.χ. βρίσκω όλες τις εναλλακτικές διαδρομές μεταξύ δύο πόλεων και επιλέγω την συντομότερη.
- Η Prolog παρέχει τη δυνατότητα "συλλογής" των λύσεων σε λίστα μέσω 3 κατηγορημάτων.
  - **bagof**, **setof** και **findall**

Κατηγόρημα findall/3

- Το **findall(Var,Goal,List)** πετυχαίνει όταν:
  - στη λίστα **List** υπάρχουν ως στοιχεία όλες οι εναλλακτικές τιμές που μπορεί να πάρει η μεταβλητή **Var.**
  - η οποία (μεταβλητή) υπάρχει μέσα στον στόχο **Goal.**
  - έτσι ώστε ο στόχος **Goal** να αληθεύει.

findall – Παράδειγμα 1/2

- Υπάρχουν τα ακόλουθα 2 γεγονότα:

father(nick,anna).

father(nick,john).

- Με ποια ερώτηση βρίσκω ένα παιδί του nick (ή όλα);

?- father(nick,X).

X = anna;

X = john;

No

- Κάθε φορά επιστρέφεται μόνο ένα παιδί και ξεχνιέται το άλλο λόγω οπισθοδρόμησης.

findall – Παράδειγμα 2/2

father(nick,anna). father(nick,john).

Βρες όλα τα παιδιά του **nick**

- Εκτελείται η ακόλουθη κλήση στην Prolog:

?- findall(X,father(nick,X),List).

List = [anna,john], X = _1

- Η μεταβλητή **List** θα έχει σε λίστα όλα τα παιδιά του **nick.**
- Η μεταβλητή **X** θα πάρει διαδοχικά όλες τις εναλλακτικές τιμές **anna** και **john.**
  - Αλλά τελικά δεν θα ενοποιηθεί με καμία από αυτές.

findall – Παράδειγμα 2

- Αν ο στόχος δεν ικανοποιείται τότε η **findall** επιστρέφει την κενή λίστα.

?- findall(X,father(john,X),List).

List = [], X = _1

Βρες όλα τα παιδιά του **john.**

- Αντί μεταβλητής, το πρώτο όρισμα της **findall** μπορεί να περιέχει <u>οποιονδήποτε</u> όρο, ο οποίος περιέχει μεταβλητές που υπάρχουν μέσα στην κλήση:

?- findall(child(X),father(nick,X),L).

L = [child(anna),child(john)], X = _1

Βρες όλα τα παιδιά του **nick.**

findall – Παράδειγμα 3

- Μέσα στην κλήση μπορεί να υπάρχουν περισσότερες από 1 μεταβλητές, οι οποίες να «συλλέγονται» στην τελική λύση.
  - Πρέπει όλες οι μεταβλητές να αναφέρονται στο 1ο όρισμα.

?- findall(child(X,Y),father(Y,X),L).

L = [child(anna,nick),child(john,nick)],

X = _1, Y = _2

Βρες όλα τα ζεύγη παιδί-πατέρας.

findall – Παράδειγμα 4

- Μέσα στην κλήση μπορεί να υπάρχουν περισσότερες από 1 μεταβλητές, οι οποίες να ***μην*** «συλλέγονται» στην τελική λύση.
  - Δεν αναφέρονται όλες οι μεταβλητές στο 1ο όρισμα.

father(nick,anna). father(nick,john).

father(george,jim).

?- findall(Y,father(Y,X),L).

L = [nick,nick,george],

X = _1, Y = _2

Βρες όλους τους πατεράδες.

findall – Παράδειγμα 5

- Η κλήση μπορεί να είναι σύνθετη, δηλαδή να περιλαμβάνει πολλούς απλούς στόχους.
  - Πρέπει οι στόχοι να περικλείονται σε ένα ζεύγος παρενθέσεων.
  - Αλλιώς το **findall** θα φαίνεται ότι έχει πάνω από 3 ορίσματα.

```
?- findall(X-Y,
           (father(Z,X),father(Z,Y),X\=Y),
           L).

L = [anna-john,john-anna],
X = _1, Y = _2, Z = _3
```

Βρες όλα τα αδέλφια.

Κατηγόρημα bagof/3

- Έχει την ίδια σύνταξη με το **findall.**

bagof(Var,Goal,List)

- Έχουν θεωρητικές διαφορές, οι οποίες όμως γίνονται αντιληπτές στο χρήστη σε 2 μόνο περιπτώσεις χρήσης.
  - Όταν δεν υπάρχει λύση.
  - Όταν υπάρχουν μεταβλητές στον στόχο, οι οποίες δεν συλλέγονται.
- Σε όλες τις υπόλοιπες περιπτώσεις η **bagof** συμπεριφέρεται όπως η **findall.**

bagof – Παράδειγμα 1

- Υπάρχουν τα ακόλουθα 2 γεγονότα:

father(nick,anna). father(nick,john).

Ίδια συμπεριφορά με findall.

- Εκτελείται η ακόλουθη κλήση στην Prolog:

?- bagof(X,father(nick,X),List).

List = [anna,john], X = _1

Βρες όλα τα παιδιά του **nick.**

- Η μεταβλητή **List** θα έχει σε λίστα όλα τα παιδιά του **nick.**
- Η μεταβλητή **X** θα πάρει διαδοχικά όλες τις εναλλακτικές τιμές **anna** και **john.**
  - Αλλά τελικά δεν θα ενοποιηθεί με καμία από αυτές.

bagof – Παράδειγμα 2

Διαφορετική συμπεριφορά με findall.

- Αν ο στόχος δεν ικανοποιείται τότε η **bagof** αποτυγχάνει.

?- bagof(X,father(john,X),List).

Βρες όλα τα παιδιά του **john.**

No

- Διαφορά με **findall.**
  - Αν ο στόχος δεν ικανοποιείται τότε η **findall** επιστρέφει την κενή λίστα.

?- findall(X,father(john,X),List).

List = [], X = _1

bagof – Παράδειγμα 3

- Μέσα στην κλήση μπορεί να υπάρχουν περισσότερες από 1 μεταβλητές, οι οποίες να ***μην*** «συλλέγονται» στην τελική λύση.
  - Δεν αναφέρονται όλες οι μεταβλητές στο 1ο όρισμα.
  - Η **bagof** δίνει ***μία*** τιμή σε κάθε τέτοια μεταβλητή και επιστρέφει ***όλες*** τις τιμές των «συλλεγόμενων» μεταβλητών.

father(nick,anna). father(nick,john).

father(george,jim).

?- bagof(X,father(Y,X),L).

Y=nick, L = [anna,john], X = _1;

Y=george, L = [jim], X = _1

Βρες όλα τα παιδιά, *κάποιου* πατέρα Υ.

Διαφορετική συμπεριφορά με findall.

bagof - findall – Διαφορά

- Η **findall** στην αντίστοιχη κλήση θεωρεί ότι μας ενδιαφέρουν ***όλες*** οι τιμές της μεταβλητής **Y.**

father(nick,anna). father(nick,john).

father(george,jim).

?- findall(X,father(Y,X),L).

L = [anna,john,jim],

Y = _2

Βρες όλα τα παιδιά, ανεξαρτήτως πατέρα.

bagof - findall – εξομοίωση

- Για να συμπεριφερθεί η **bagof** όπως η **findall** πρέπει οι μεταβλητές που δεν «συλλέγονται» στην λύση να δηλώνονται ως «καθολικές».

father(nick,anna). father(nick,john).

father(george,jim).

?- bagof(X,Y^father(Y,X),L).

L = [anna,john,jim],

Y = _2

Βρες όλα τα παιδιά, ανεξαρτήτως πατέρα (**Για κάθε ()** τιμή του Υ)

Κατηγόρημα setof

- Έχει την ίδια σύνταξη με το **bagof**

setof(Var,Goal,List)

- Διαφορές **setof** - **bagof**
  - Η **setof** απομακρύνει τα διπλά στοιχεία από τη λίστα
  - Η **setof** διατάσσει τα στοιχεία στην λίστα σε αύξουσα σειρά

setof – Παράδειγμα 1

- Υπάρχουν τα ακόλουθα 2 γεγονότα:

father(nick,john).

father(nick,anna).

- Αν χρησιμοποιηθεί **bagof**:

?- bagof(X,father(nick,X),List).

List = [john,anna], X = _1

- Αν χρησιμοποιηθεί **setof**:

?- setof(X,father(nick,X),List).

List = [anna,john], X = _1

Βρες όλα τα παιδιά του **nick.**

setof – Παράδειγμα 2

- Υπάρχουν τα ακόλουθα 3 γεγονότα:

father(nick,anna). father(nick,john).

father(george,jim).

- Αν χρησιμοποιηθεί **bagof**:

?- bagof(Y,X^father(Y,X),L).

L = [nick,nick,george], X = _1, Y = _2

Βρες όλους τους πατεράδες.

- Αν χρησιμοποιηθεί **setof**:

?- setof(Y,X^father(Y,X),L).

L = [george,nick], X = _1, Y = _2

Άσκηση findall

- Έστω τα ακόλουθα γεγονότα:

next(greece,turkey). **next(greece,albania).**

next(greece,bulgaria). **next(greece,fyrom).**

next(bulgaria,romania). **…**

- Να γραφεί κατηγόρημα που να υπολογίζει με πόσες χώρες συνορεύει μια συγκεκριμένη χώρα

```
count(Country,T) :-
        findall(N,next(Country,N),Neighbours),
        length(Neighbours,T).
```

Άσκηση count

?– count(greece,X).

X = 4

Η κλήση ***next(greece,N)*** *έχει 4 εναλλακτικές απαντήσεις.*

? – count(Country,X).

Country = _

X = 30

Country*: μεταβλητή χωρίς τιμή.*

Η κλήση ***next(Country,N)*** *έχει 30 εναλλακτικές απαντήσεις (όλα τα γεγονότα* ***next****).*

Παράδειγμα count με χρήση bagof

```
count(Country,T) :-
        bagof(N,next(Country,N),Neighbours),
        length(Neighbours,T).

?– count(greece,X).
X = 4

?– count(Country,X).
Country = ‘Greece’
X = 4 ;
Country = ‘Turkey’
X = 5 ;
```

(ίδια απάντηση με findall)

*Εξαιτίας του **bagof** η μεταβλητή **Country** πρώτα παίρνει κάποια συγκεκριμένη τιμή και μετά αναζητούνται όλα τα **N** που συνορεύουν με αυτήν, ενώ στη **findall,** η **Country** είναι ‘αδιάφορη’ μεταβλητή, χωρίς τιμή.*

Αντιμεταθετική Ιδιότητα (1/2)

- Αν θέλαμε να ενσωματώσουμε την αντιμεταθετική ιδιότητα στο προηγούμενο παράδειγμα.
- **1ος τρόπος**: Προσθέτω αντιμεταθετικό κατηγόρημα **next_to**

```
next_to(X,Y) :- next(X,Y).

next_to(X,Y) :- next(Y,X).
```

- Πρέπει να αλλάξει και η κλήση μέσα στο **findall**

```
count(Country,T) :-
        findall(N,next_to(Country,N),Neighbours),
        length(Neighbours,T).
```

Αντιμεταθετική Ιδιότητα (2/2)

- **2ος τρόπος**: Προσθέτω διάζευξη κλήσεων στο **findall**

```
count(Country,T) :-
        findall(N,
                (next(Country,N); next(N,Country)),
                Neighbours),
        length(Neighbours,T).
```

- Η διάζευξη είναι σύνθετη κλήση και πρέπει να μπει σε παρενθέσεις.

Άσκηση findall, setof

- Να ορισθεί το κατηγόρημα **employees(Dept,Employees)** το οποίο να επιστρέφει όλους τους υπαλλήλους (λίστα **Employees)** που εργάζονται στο τμήμα **Dept.**

```
employees(Dept,Employees):-
    findall(Emp,
            employee(_,data(Emp,_,_,_),department(Dept))),
            Employees).
```

*Η λίστα **Employees** περιέχει τις τιμές που παίρνει η μεταβλητή **Emp***

*Η λίστα στην οποία "μαζεύονται" οι τιμές της μεταβλητής **Emp***

*Κλήση η οποία "παράγει" τις τιμές για τη μεταβλητή **Emp***

Γεγονότα υπαλλήλων

```
employee(   4444,
            data(   name(   first(nick),
                            last(bassiliades),
                            middle(dimitrios)),
                    id(     number(     letter(n),
                                        no(355324)),
                            authority(  at(a),
                                        city(thessaloniki)),
                            date(23,6,1983)),
                    address( road(tsimiski),
                             number(4),
                             city(thessaloniki)),
                    hire_date(date(10,10,1999)),
                    department(informatics)
```

number(n,355324)

at(a,thessaloniki)

hire_date(
day(10),
month(10),
year(1999))

Άσκηση (συνέχεια)

- Να ορισθεί κατηγόρημα **no_of_employees(Dept,N)** το οποίο να επιστρέφει το συνολικό αριθμό υπαλλήλων που εργάζονται στο τμήμα **Dept.**

no_of_employees(Dept,N):-

employees(Dept,Employees),

length(Employees,N).

*Λίστα όλων των υπαλλήλων που δουλεύουν στο τμήμα **Dept.***

Εύρεση μήκους της λίστας - δηλαδή του αριθμού των υπαλλήλων.

Άσκηση (setof)

- Να ορισθεί το κατηγόρημα **print_employees(Dept),** το οποίο να τυπώνει τους υπαλλήλους του τμήματος **Dept** στην οθόνη, ταξινομη-μένους κατά επίθετο, όνομα, πατρώνυμο (1 υπάλληλο ανά γραμμή).

```
print_employees(Dept):-
    setof(L-F-M,
          C^ID^Ad^HD^
              employee(C,
                       data(name(first(F),last(L),middle(M)),
                            ID,Ad,HD,department(Dept))),
          Employees),
    print_employees_aux(Employees).
```

*Η λίστα **Employees** θα περιέχει τους σύνθετους όρους που δημιουργούνται από το συνδυασμό των τιμών που παίρνουν οι 3 μεταβλητές.*

Δεν μας ενδιαφέρουν οι τιμές των 4 αυτών μεταβλητών, συνεπώς πρέπει να δηλώσουμε ότι είναι "ελεύθερες".

Η λίστα στην οποία "μαζεύονται" οι τιμές.

*Κλήση η οποία "παράγει" τις τιμές. Γίνεται ανάλυση του ονόματος στα στοιχεία του, γιατί πρέπει να γίνει αλλαγή της σειράς του **L** με το **F.***

Άσκηση - συνέχεια

- Βοηθητικό κατηγόρημα για την εκτύπωση των στοιχείων της λίστας.

```
print_employees_aux([]).
print_employees_aux([L-F-M|Tail]) :-
    write(F), write(' '),     % Κενό μεταξύ των λέξεων.
    write(M), write(' '),
    write(L), nl,             % Αλλαγή σειράς - Απαραίτητη!
    print_employees_aux(Tail).
```

Εκτύπωση των υπολοίπων στοιχείων της λίστας.

Ασκήσεις – findall (1/2)

- Δίνεται μια βάση με γεωγραφικά στοιχεία πληθυσμού (**population)** και έκτασης (**area**) διαφόρων κρατών, καθώς επίσης και πληροφορίες για την ήπειρο που ανήκει κάθε χώρα.
- Να ορισθεί η σχέση **density** που να επιστρέφει την πυκνότητα του πληθυσμού μιας **ηπείρου**.

population(usa,203). **area(usa,3).**

population(india,750). **area(india,1).**

population(china,1200). **area(china,4).**

population(brazil,112). **area(brazil,3).**

…

belongs_to(brazil,america). **belongs_to(usa,america).**

…

Ασκήσεις – findall (2/2)

```
density(Continent,D) :-
        findall(P,      (belongs_to(Country,Continent),
                         population(Country,P)),
                Pops),
        findall(A,      (belongs_to(Country,Continent),
                         area(Country,A)),
                Areas),
        sumlist(Pops,Population),
        sumlist(Areas,Area),
        if_then_else(Area>0, D is Population / Area,
                             D = error).
```

Πρόγραμμα Διαχείρισης Γεγονότων

- Το ακόλουθο πρόγραμμα διαχειρίζεται γεγονότα βαθμολογίας φοιτητών, φορτώνοντάς τα από αρχείο δυναμικά στη μνήμη.
- Υπάρχει κεντρικό μενού επιλογών για τις διάφορες λειτουργίες του προγράμματος.
- Στο πρόγραμμα συγκεντρώνονται πολλές τεχνικές που έχουν παρουσιαστεί ως τώρα.
  - Επανάληψη μέσω οπισθοδρόμησης, κατηγορήματα διαχείρισης λύσεων (findall/setof), δυναμική τροποποίηση προγράμματος (assert/retract), κλπ.

Κεντρικό μενού

```
run :-
    write('Main Menu:'), nl, nl,
    write('1 - Load a student table'), nl,
    write('2 - Save a student table'), nl,
    write('3 - Find a student grade'), nl,
    write('4 - Insert a student grade'), nl,
    write('5 - Delete a student grade'), nl,
    write('6 - Find how many student have passed'), nl,
    write('7 - Average Class Grade'), nl,
    write('8 - Print Student List (ordered)'), nl,
    write('0 - Exit'), nl, nl,
    write('Choice: '),
    read(Answer),
    do_on_choice(Answer).
```

Ανάγνωση επιλογής χρήστη.

Εκτέλεση αντίστοιχης λειτουργίας.

Φόρτωμα γεγονότων από αρχείο

```
do_on_choice(1) :- !,  % Η αποκοπή κόβει τις άλλες επιλογές.
    nl, write('File name: '), read(File),
    see(File),                          % Άνοιγμα αρχείου για ανάγνωση.
    repeat,                             % Επανάληψη μέσω οπισθοδρόμησης.
    read(Fact),                 % Ανάγνωση 1 γεγονότος από αρχείο.
    ins_fact(Fact),             % Εισαγωγή γεγονότος στη βάση.
    % Αποτυχία και επιστροφή στο repeat, ή επιτυχία και συνέχεια.
    Fact == end_of_file, !,     % Αποκοπή για το repeat.
    nl, write('File loaded.'), nl,
    seen,                       % Κλείσιμο αρχείου.
    run.                        % Εκτέλεση του μενού (επιστροφή).
ins_fact(end_of_file) :- !.             % Τέλος αρχείου.
ins_fact(Fact) :- assert(Fact).     % Εισαγωγή γεγονότος.
```

Αποθήκευση γεγονότων σε αρχείο

```
do_on_choice(2) :- !,
    write('File name: '), read(Name),
    tell(Name),                    % Άνοιγμα αρχείου για εγγραφή.
    save_facts,                    % Αποθήκευση γεγονότων.
    told,                          % Κλείσιμο αρχείου.
    nl, write('File saved.'), nl,
    run.                  % Εκτέλεση του μενού (επιστροφή).
save_facts :-
    student(Name,Grade),                % Ανάκληση ενός γεγονότος.
    write(student(Name,Grade)),    % Εγγραφή στο αρχείο.
    write('.'), nl,                % Εγγραφή "τελείας"..
    fail.          % Αποτυχία και επιστροφή στο επόμενο γεγονός.
save_facts. % Όταν δεν υπάρχουν άλλα γεγονότα, απλά επιτυγχάνει.
```

Αναζήτηση γεγονότος

```
do_on_choice(3) :- !,
    nl, write('Student Name: '),
    read(Name), nl,          % Αναζήτηση βαθμού συγκεκριμένου .
    find_student(Name),      % φοιτητή βάσει του ονόματος.
    nl, run.

find_student(Name) :-        % Αν ο φοιτητής υπάρχει,
    student(Name,Grade), !,  % τότε τύπωσε το βαθμό του.
    write('Grade: '),
    write(Grade), nl.
find_student(_) :-           % Αν όχι, τύπωσε ότι δε βρέθηκε.
    write('Not found!'), nl.
```

Εισαγωγή νέου γεγονότος στη μνήμη

```
do_on_choice(4) :- !,
    nl, write('Student Name: '),
    read(Name), nl, % Ανάγνωση από το χρήστη ονόματος.
    write('Grade: '), % και βαθμού του φοιτητή.
    read(Grade), nl,
    insert_student(Name,Grade), % Εισαγωγή φοιτητή στη μνήμη.
    nl, run.
insert_student(Name,Grade) :-        % Αν ο φοιτητής υπάρχει.
    student(Name,_), !,              % (με οποιονδήποτε βαθμό).
    write('Student '), write(Name),  % μην τον ξαναπροσθέτεις.
    write(' already exists!'), nl.
insert_student(Name,Grade) :-        % Αν όχι, τότε
    assert(student(Name,Grade)).     % εισαγωγή γεγονότος.
```

Διαγραφή γεγονότος από τη μνήμη

```
do_on_choice(5) :- !,
    nl, write('Student Name: '),
    read(Name), nl,        % Ανάγνωση του ονόματος του φοιτητή.
    delete_student(Name), % Διαγραφή γεγονότος.
    nl, run.


% Διαγραφή γεγονότος βάσει ονόματος.
delete_student(Name) :-        % Αν ο φοιτητής υπάρχει, τότε διέγραψε.
    retract(student(Name,_)), !. % το γεγονός (βαθμός αδιάφορος).
delete_student(_) :-       % Αν όχι, τύπωσε ότι δε βρέθηκε.
    write('Not found!'), nl.
```

Πόσοι φοιτητές "πέρασαν" το μάθημα;

```
do_on_choice(6) :- !,
    % Μαζεύει όλους τους φοιτητές που έχουν βαθμό από 5.
    % και πάνω σε μια λίστα.
    findall(S,(student(S,G), G >= 5),L),
    length(L,N),          % Μετράει το μήκος της λίστας.
    nl, write(N), write(' students have passed!'),
    nl, nl, run.
```

Μέσος όρος βαθμολογίας τάξης

```
do_on_choice(7) :- !,
    % Μαζεύει όλους τους βαθμούς των φοιτητών σε μια λίστα.
    findall(G,student(_,G),L),
    % Βρίσκει το μέσο όρο των αριθμών της λίστας.
    average_list(L,Avg),
    nl, write('Course Average: '), write(Avg), nl, nl,
    run.

average_list([],'None student in the DB!') :- !.
average_list(L,A) :-
    sumlist(L,S), length(L,N), A is S / N.
```

Εκτύπωση λίστας φοιτητών

Ταξινόμηση πρώτα στο βαθμό και μετά στο όνομα

```
do_on_choice(8) :- !,
    % Μαζεύει όλους τους φοιτητές και τους βαθμούς σε μια λίστα
    % Στην ταξινόμηση προτάσσεται ο βαθμός (μπαίνει πρώτος).
    setof(G-S,student(S,G),L),
    print_students(L),          % Τυπώνει τη λίστα.
    nl, run.

                                % Δεν υπάρχουν άλλοι φοιτητές -
print_students([]) :- nl.       % τύπωσε κενή γραμμή και σταμάτα.
print_students([G-S|T]) :-          % Τύπωσε τον πρώτο φοιτητή
    write(S), write(' : '), write(G), nl, % και το βαθμό του
    print_students(T).    % και μετά αναδρομικά τους υπόλοιπους.
```

Πρόγραμμα διαχείρισης γεγονότων

- Έξοδος από το μενού και τερματισμός προγράμματος:

```
do_on_choice(0) :- !,
    nl, write('Goodbye!'), nl.
```

- Εκτύπωση μηνύματος αν εισαχθεί λάθος επιλογή.
  - Ο κανόνας είναι πάντα τελευταίος και εκτελείται μόνο αν αποτύχουν όλοι οι προηγούμενοι.

```
do_on_choice(_) :-
    nl, write('Give a number between 0 and 8!'), nl,
    run.
```

Σημείωμα Αναφοράς

Σημείωμα Αδειοδότησης

Τέλος ενότητας

Επεξεργασία: Εμμανουήλ Ρήγας

Θεσσαλονίκη, Εαρινό Εξάμηνο 2013-2014

Ευρωπαϊκή Ένωση
Ευρωπαϊκό Κοινωνικό Ταμείο
ΕΠΙΧΕΙΡΗΣΙΑΚΟ ΠΡΟΓΡΑΜΜΑ
ΕΚΠΑΙΔΕΥΣΗ ΚΑΙ ΔΙΑ ΒΙΟΥ ΜΑΘΗΣΗ
επένδυση στην κοινωνία της γνώσης
ΥΠΟΥΡΓΕΙΟ ΠΑΙΔΕΙΑΣ ΚΑΙ ΘΡΗΣΚΕΥΜΑΤΩΝ
ΕΙΔΙΚΗ ΥΠΗΡΕΣΙΑ ΔΙΑΧΕΙΡΙΣΗΣ
ΕΣΠΑ 2007-2013
πρόγραμμα για την ανάπτυξη
ΕΥΡΩΠΑΪΚΟ ΚΟΙΝΩΝΙΚΟ ΤΑΜΕΙΟ
Με τη συγχρηματοδότηση της Ελλάδας και της Ευρωπαϊκής Ένωσης

Σημειώματα

Διατήρηση Σημειωμάτων

Οποιαδήποτε αναπαραγωγή ή διασκευή του υλικού θα πρέπει να συμπεριλαμβάνει:

- το Σημείωμα Αναφοράς
- το Σημείωμα Αδειοδότησης
- τη δήλωση Διατήρησης Σημειωμάτων
- το Σημείωμα Χρήσης Έργων Τρίτων (εφόσον υπάρχει)

μαζί με τους συνοδευόμενους υπερσυνδέσμους.