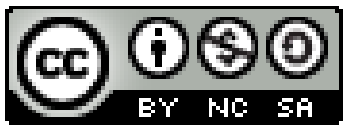
ΑΡΙΣΤΟΤΕΛΕΙΟ
ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΟΝΙΚΗΣ

ΑΝΟΙΚΤΑ
ΑΚΑΔΗΜΑΪΚΑ
ΜΑΘΗΜΑΤΑ

Υπολογιστική Λογική και Λογικός Προγραμματισμός

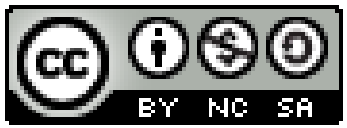
Ενότητα 6: Γλώσσα Prolog: Ενσωματωμένα Κατηγορήματα, Αριθμητικές Διαδικασίες, Διαδικασίες Εισόδου-Εξόδου, Χειρισμός Συμβολοσειρών

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.

Γλώσσα Prolog: Ενσωματωμένα Κατηγορήματα, Αριθμητικές Διαδικασίες, Διαδικασίες Εισόδου-Εξόδου, Χειρισμός Συμβολοσειρών.

Ενσωματωμένα Κατηγορήματα

- Η Prolog, όπως παρουσιάστηκε μέχρι τώρα, είναι απόλυτα συμβατή με τον καθαρό λογικό προγραμματισμό.
  - υλοποίηση σε υπολογιστή ενός διερμηνέα των προτάσεων Horn.
- Όμως η Prolog χρησιμοποιείται και ως πρακτική γλώσσα προγραμματισμού γενικού σκοπού
  - εκτός από ένα εργαλείο απόδειξης θεωρημάτων.
- Είναι απαραίτητη η επέκτασή της με δυνατότητες που βρίσκονται πέρα (extra-logical) και πάνω (meta-logical) από τη λογική.
- Οι δυνατότητες αυτές παίρνουν τη μορφή των ενσωματωμένων κατηγορημάτων (built-in predicates).
  - Παρέχονται από τον εκάστοτε κατασκευαστή της έκδοσης του διερμηνέα της Prolog.

Ενσωματωμένα Κατηγορήματα

- Αφορούν:
  - Μαθηματικές πράξεις και συναρτήσεις.
  - Είσοδος/έξοδος σε αρχεία ή/και συσκευές.
  - Έλεγχος τύπου δεδομένων.
  - Έλεγχος εκτέλεσης προγράμματος.
  - Διαχείριση προγράμματος.
  - Διαχείριση λύσεων.
- Θα παρουσιαστούν ενδεικτικά κάποια από τα πιο συνηθισμένα ενσωματωμένα κατηγορήματα.
  - Περισσότερα στα εγχειρίδια χρήσης.

Ενοποίηση και Σύγκριση Όρων

- Η Prolog κάνει ενοποίηση όρων κατά την προσπάθεια ικανοποίησης ενός στόχου.
- Υπάρχει η δυνατότητα η ενοποίηση όρων να γίνει με σαφή κλήση του ενσωματωμένου κατηγορήματος "=" στο σώμα ενός κανόνα, ή σε μια σύνθετη ερώτηση.
- Το κατηγόρημα αυτό αληθεύει αν οι δύο όροι μπορούν να ταυτοποιηθούν.
  - Αν ναι, στη συνέχεια ταυτοποιούνται.

Παραδείγματα Ενοποίησης

?- X = 5.

X = 5

?- X=Y.

X = Y = _

?- f(a,b) = f(a,b).

Yes

?- f(X,b) = f(a,Y).

X=a, Y=b

?- f(a) = g(a).

No

?- X=5, Y=6, X=Y.

no

?- X=5, X=Y.

X=5, Y=5

?- X=Y, X=5.

X=5, Y=5

?- X=Y, X=5, Y=6.

no

Αντι-ταυτοποίηση

- Υπάρχει και το αντίθετο κατηγόρημα "**\=**", το οποίο αληθεύει αν οι δύο όροι δεν μπορούν να ταυτοποιηθούν.

?- X \= 5.

no

?- X \= Y.

no

?- f(a,b) \= f(a,b).

no

?- f(X,b) \= f(a,Y).

no

?- f(a) \= g(a).

yes

Κατηγορήματα Σύγκρισης Όρων

- Το κατηγόρημα "==" αληθεύει αν οι δύο όροι που συγκρίνει είναι ταυτόσημοι,
  - Έχουν την ίδια δομή και όλοι οι όροι που περιέχουν (αν είναι σύνθετοι) είναι ίδιοι.

```
?- 2==2.
yes
?- 2==3.
no
```

```
?- f(a)==f(a).
yes
?- f(X)==f(a).
no
```

```
?- X==2.
no
?- X==Y.
no
```

Παρατηρήσεις

- Μια μεταβλητή δεν είναι ταυτόσημη με μία σταθερά,
  - Η πρώτη αντιπροσωπεύει μία θέση μνήμης, ενώ η δεύτερη είναι ένα σύμβολο.
- Δύο διαφορετικές μεταβλητές δεν είναι ταυτόσημες,
  - Αντιπροσωπεύουν δύο ξεχωριστές "θέσεις μνήμης".
- Αν έχει προηγηθεί ταυτοποίηση πριν από την σύγκριση των όρων, τότε τα παραπάνω ΔΕΝ ισχύουν,
  - Η ταυτοποίηση διαφοροποιεί τη συμπεριφορά των μεταβλητών.

?- X=2, X==2.

X=2

?- X=Y, X==Y.

X = Y = _

Κατηγορήματα Σύγκρισης Όρων

- Το κατηγόρημα "\==" αληθεύει αν οι δύο όροι που συγκρίνει δεν είναι ταυτόσημοι.

?- 2 \== 3.

yes

?- 2 \==2.

no

?- X \== 2.

yes

?- X \== Y.

yes

Μαθηματικές Διαδικασίες και Συναρτήσεις

- Υπάρχει η δυνατότητα χειρισμού αριθμών και εκτέλεσης πράξεων με τη χρήση ενσωματωμένων κατηγορημάτων.
- Το πιο βασικό κατηγόρημα είναι το **is**
  - αποτιμώνται αριθμητικές εκφράσεις.
  - η τιμή τους ενοποιείται με κάποια μεταβλητή.
- Το κατηγόρημα χρησιμοποιείται με τη μορφή **X is Y.**
  - **X** είναι μια μεταβλητή ή ένας αριθμός.
  - **Y** είναι μια μαθηματική έκφραση η οποία μπορεί να περιέχει πράξεις και συναρτήσεις.
  - Αφού αποτιμηθεί η έκφραση **Y**, γίνεται προσπάθεια ενοποίησης με τον όρο **X**.

Παραδείγματα χρήσης is

```
?- X is 3 + 4.
X=7
?- 9 is 3 * 3.
yes
?- 18 is 5 - 3.
no
?- X is Y + 1.
ERROR: is/2: Arguments are
       not sufficiently instantiated
?- Y is 2, X is Y + 1.
Y=2, X=3
```

Απόδοση τιμής σε κάποια μεταβλητή κάνοντας κάποιον μαθηματικό υπολογισμό.

Έλεγχος αν η τιμή κάποιας μαθηματικής έκφρασης ισούται με κάποιον αριθμό.

Στη μαθηματική έκφραση μπορούν να χρησιμοποιηθούν μεταβλητές μόνο αν έχουν πάρει προηγουμένως κάποια τιμή.

Ενσωματωμένες Συναρτήσεις

- Υπάρχουν και ενσωματωμένες συναρτήσεις (π.χ. ημίτονο, τετραγωνική ρίζα, κλπ) οι οποίες διαφέρουν από έκδοση σε έκδοση της γλώσσας.

?- Y is sqrt(9).

Y = 3

- Άλλες:
  - **cos(N)** - συν(N)
  - **sin(N)** - ημ(N)
  - **tan(N)** - εφ(N)
  - **exp(N)** - e^N

Μαθηματικές Ισότητες/Ανισότητες

- Εκτός από το **is** υπάρχουν και τα κατηγορήματα "**=:=**" (ισότητα) και "**=\=**" (ανισότητα).
  - Εκτελούν πράξεις και από τις δυο πλευρές της ισότητας/ανισότητας.

?- 1 + 2 =:= 2 + 1.

yes

?- 5 – 3 =:= 2 + 2.

no

?- 5 – 3 =\= 2 + 2.

yes

*Ισχύουν τα ίδια που ισχύουν για το **is**, όσον αφορά τις μεταβλητές.*

Παρατηρήσεις (1/2)

- Αριθμητικές πράξεις μπορούμε να κάνουμε μόνο με την χρήση των **is**, **=:=**, **=\=**
  - Σε οποιαδήποτε άλλη περίπτωση οι αριθμητικές εκφράσεις είναι σύνθετοι όροι.

?- X is 1+2.

X=3

?- X = 1+2.

X=1+2

?- X == 1+2.

no

?- X =:= 1+2.

ERROR: …

Παρατηρήσεις (2/2)

?- 1+2 =:= 2+1.	?- 1+2 = 2+1.	?- 1+2 == 2+1.
yes	no	no
?- 1+2 =:= 1+2.	?- 1+2 = 1+2.	?- 1+2 == 1+2.
yes	yes	yes
?- 1+A=B+2.	?- 1+A=:=B+2.	?- 1+A==B+2.
A=2, B=1	ERROR…	no

Ασκήσεις - Αριθμητικές Διαδικασίες

- Δίνεται μια βάση με γεωγραφικά στοιχεία πληθυσμού (**population)** και έκτασης (area) διαφόρων περιοχών.
- Να ορισθεί η σχέση **density** που να επιστρέφει την πυκνότητα του πληθυσμού, δηλαδή τον αριθμό των κατοίκων μιας περιοχής ανά τετραγωνικό χιλιόμετρο.

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

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

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

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

density(X,Y) :-

population(X,P), area(X,A), Y is P / A.

Κατηγορήματα Σύγκρισης

- Η μοναδική ιδιομορφία της Prolog είναι ότι ο τελεστής "*μικρότερο ή ίσο*" γράφεται ανάποδα "=<", για να μη θυμίζει "βέλος".

?- 5 > 3.

yes

?- 5 =< 3

no.

?- X > 3.

Error...

?- 5+3 > 2+1.

yes

?- X is 7, X > 3.

yes

?- X=7, Y=3, X > Y+1.

yes

Οι εκφράσεις αριστερά και δεξιά των τελεστών σύγκρισης μπορούν να είναι μαθηματικές εκφράσεις, οι οποίες πρώτα αποτιμώνται και μετά γίνεται η σύγκριση.

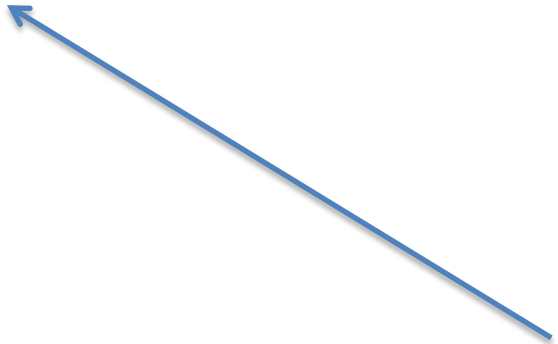
Ασκήσεις - Αριθμητικές Διαδικασίες

- Να ορισθεί η σχέση **sign** (πρόσημο) που να επιστρέφει την ένδειξη positive, negative, zero, ανάλογα με την τιμή της παραμέτρου.

sign(X, positive) :- X > 0.

sign(X, zero) :- X =:= 0.

sign(X, negative) :- X<0.



?- sign(5,A).

A=positive

?- sign(5-5,A). *% Οφείλεται στη χρήση του =:=*

A=zero

?- sign(5-7,A). *% Οφείλεται στη χρήση του <*

A=negative

Κατηγόρημα sorted/1

- Επιτυγχάνει εάν η λίστα που δέχεται σαν όρισμα εισόδου είναι ταξινομημένη σε αύξουσα σειρά, αλλιώς αποτυγχάνει.
- α) εάν η λίστα έχει 1 ή κανένα στοιχείο, τότε είναι ταξινομημένη.
- β) εάν η λίστα έχει 2 ή περισσότερα στοιχεία τότε πρέπει:
  - β1) το πρώτο στοιχείο της λίστας να είναι μικρότερο ή ίσο από το δεύτερο.
  - β2) η ουρά της λίστας να είναι ταξινομημένη.

```
sorted([]).
sorted([_]).
sorted([X,Y|List]) :-
    X=<Y, sorted([Y|List]) .
```

```
?- sorted([1,2,5,9]).
yes
?- sorted([2,1,5,9]).
no
?- sorted([1,5,2,9]).
no
```

Μέγιστο Στοιχείο Λίστας

- α) εάν η λίστα έχει ένα στοιχείο, τότε αυτό είναι το μέγιστο.
- β) εάν η λίστα έχει δύο ή περισσότερα στοιχεία τότε:
  - β1) βρίσκει το μέγιστο στοιχείο της ουράς της.
  - β2) συγκρίνει το μέγιστο στοιχείο της ουράς με την κεφαλή της λίστας και επιστρέφει τη μέγιστη από τις δύο τιμές.
- Χρησιμοποιείται το κατηγόρημα **max/3.**
  - Δέχεται 2 τιμές σαν είσοδο στα 2 πρώτα ορίσματα και επιστρέφει στο 3ο όρισμα την μεγαλύτερη από τις 2.

```
list_max([X],X).
list_max([X|Tail],M) :-
    list_max(Tail,Ml),
    max(X,Ml,M).
```

```
max(X,Y,X) :- X>Y.
max(X,Y,Y) :- X=<Y.
```

Παράδειγμα εκτέλεσης

```
?- list_max([2,3,7,1,5],A).

A=7

?- list_max([2,3,7,1,5],7).

yes

?- list_max([2,3,7,1,5],5).

no
```

```
list_max([X],X).
list_max([X|Tail],M) :-
        list_max(Tail,Ml),
        max(X,Ml,M).
```

```
max(X,Y,X) :- X>Y.
max(X,Y,Y) :- X=<Y.
```

Μέγιστος Κοινός Διαιρέτης

- εάν οι 2 αριθμοί είναι ίσοι, τότε ο ΜΚΔ είναι οι ίδιοι οι αριθμοί.
- εάν οι 2 αριθμοί δεν είναι ίσοι, αφαίρεσε από τον μεγαλύτερο τον μικρότερο και υπολόγισε τον ΜΚΔ του υπόλοιπου και του αφαιρετέου.
  - Αυτός είναι ο ΜΚΔ και των 2 αρχικών αριθμών.

```
gcd(X,X,X).
gcd(X,Y,D) :-
    X<Y,
    Y1 is Y-X,
    gcd(X,Y1,D).
gcd(X,Y,D) :-
    X1 is X - Y,
    gcd(X1,Y,D).
```

?- gcd(4,12,D).

D=4

?- gcd(4,12,D).

↓

?- gcd(4,8,D).

↓

?- gcd(4,4,D).

Άθροισμα Στοιχείων Λίστας (1/2)

- Να γραφεί διαδικασία, η οποία να υπολογίζει το άθροισμα των στοιχείων μιας λίστας.
- Εάν η λίστα είναι κενή, το άθροισμα των στοιχείων της είναι μηδέν.
- Εάν η λίστα δεν είναι κενή, τότε
  - υπολόγισε το άθροισμα των στοιχείων της ουράς της,
  - πρόσθεσε στο αποτέλεσμα την τιμή της κεφαλής της
  - και επέστρεψε το τελευταίο αποτέλεσμα.

Άθροισμα Στοιχείων Λίστας (2/2)

- Καθαρά αναδρομικός ορισμός.

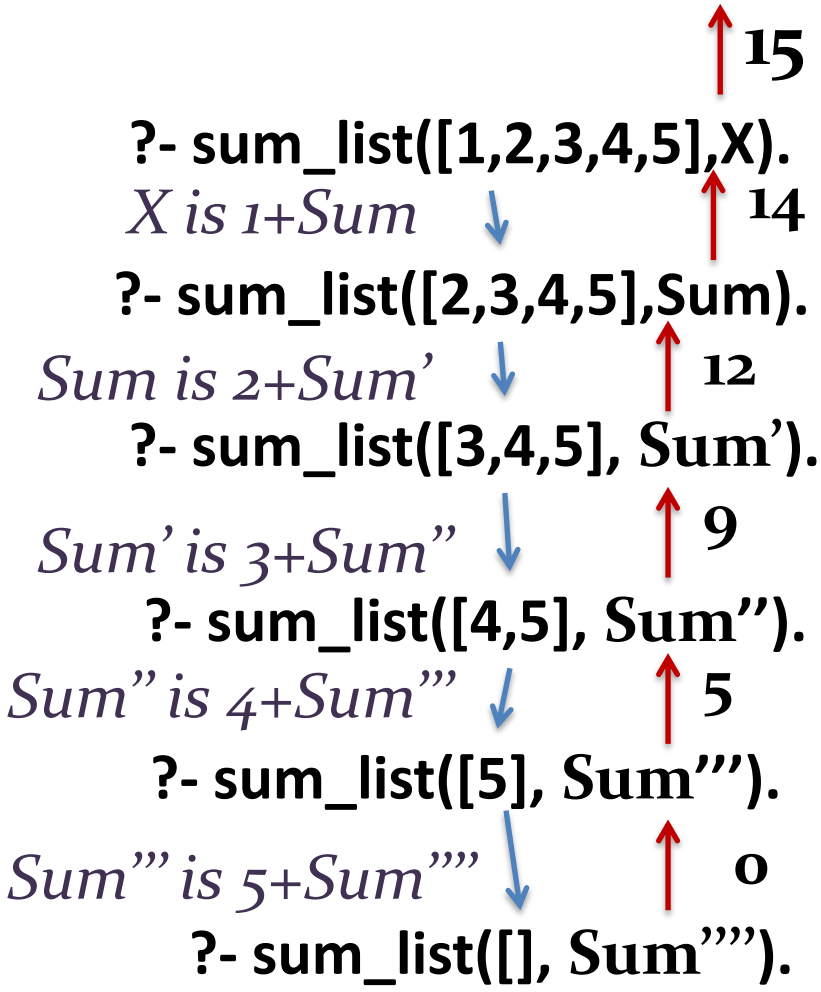
```
sum_list([],0).
sum_list([H|T], Sum) :-
        sum_list(T, TempSum),
        Sum is H + TempSum.
```

```
?- sum_list([1,2,3,4,5],X).
X=15
```

*Η άθροιση πρέπει να γίνει **ΜΕΤΑ** την αναδρομή, γιατί αλλιώς η μεταβλητή TempSum δεν θα έχει τιμή και θα προκληθεί λάθος.*

Εκτέλεση

```
sum_list([],0).
sum_list([H|T], Sum) :-
    sum_list(T, TempSum),
    Sum is H + TempSum.
```



Λάθη που γίνονται συχνά (1/3)

```
sum_list([],0).
sum_list([H|T], Sum) :-
        Sum is H + TempSum,
        sum_list(T, TempSum).
```

- Η πράξη δεν μπορεί να εκτελεστεί γιατί το **TempSum** την στιγμή της κλήσης **δεν έχει τιμή**.

Λάθη που γίνονται συχνά (2/3)

```
sum_list([],0).
sum_list([H|T], Sum) :-
        sum_list(T, TempSum),
        TempSum is Sum - H.
```

- Η πράξη αυτή δεν μπορεί να εκτελεστεί γιατί το **Sum** την στιγμή της κλήσης **δεν έχει τιμή**.
  - Έξοδος της διαδικασίας θεωρείται το **Sum**, επομένως αυτό πρέπει να υπολογιστεί στο αριστερό μέρος της **is**.

Λάθη που γίνονται συχνά (3/3)

```
sum_list([],0).
sum_list([H|T], Sum) :-
        sum_list(T, Sum),
        Sum is Sum + H.
```

- Η πράξη αυτή **αποτυγχάνει** γιατί το **Sum** δεν μπορεί να αλλάξει τιμή από την στιγμή που έχει πάρει κάποια.
  - Στην Prolog οι μεταβλητές είναι λογικές/μαθηματικές και όχι προγραμματιστικές (θέσεις μνήμης).

Άθροισμα Στοιχείων Λίστας – Επαναληπτικός ορισμός

- Με τη χρήση βοηθητικού κατηγορήματος
  - Ονομάζεται έτσι γιατί διαδικαστικά θυμίζει την επανάληψη των κλασικών γλωσσών προγραμματισμού.

```
sum_list(List,Sum) :-
        sum_list_aux(List,0,Sum).

sum_list_aux([],Sum,Sum).
sum_list_aux([H|T],Temp,Sum) :-
        Next is Temp + H,
        sum_list_aux(T,Next,Sum).
```

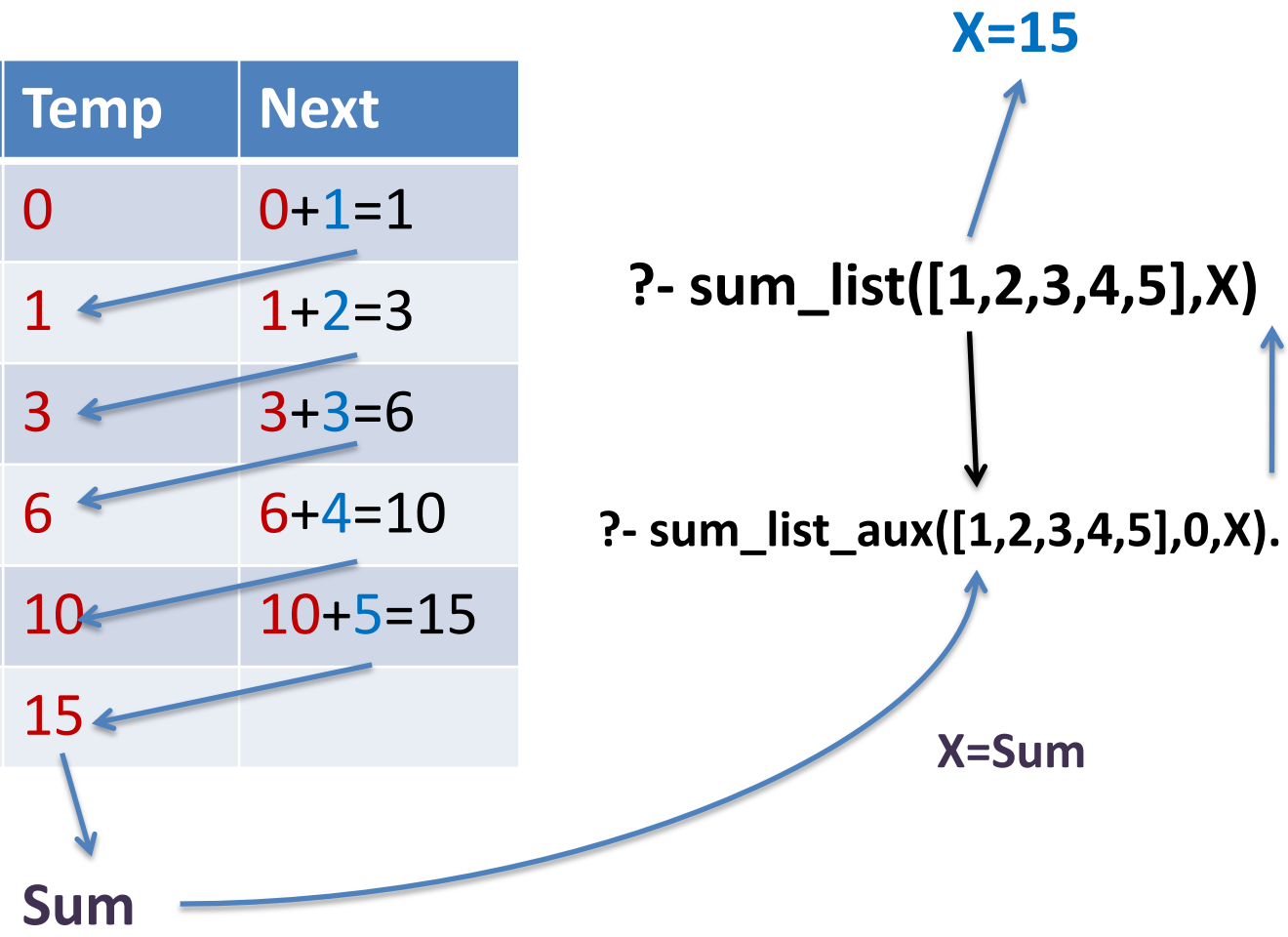
Θέτουμε μία αρχική τιμή.

Σε κάθε επανάληψη υπολογίζουμε την επόμενη προσωρινή τιμή με βάση την προηγούμενη προσωρινή τιμή (ξεκινώντας από την αρχική).

Όταν αληθεύει η συνθήκη τερματισμού, τότε η προσωρινή τιμή είναι και το τελικό αποτέλεσμα.

Εκτέλεση

[H|T]	Temp	Next
[1,2,3,4,5]	0	0+1=1
[2,3,4,5]	1	1+2=3
[3,4,5]	3	3+3=6
[4,5]	6	6+4=10
[5]	10	10+5=15
[]	15	



Επανάληψη vs. Αναδρομή

- Οι επαναληπτικές διαδικασίες υλοποιούνται με κανονική αναδρομή της Prolog.
- Δεν εισάγεται καμία καινούργια έννοια, απλά αλλάζει λίγο ο τρόπος σκέψης και προγραμματισμού.
- Χρησιμοποιούμε βοηθητικό κατηγόρημα και βοηθητικές παραμέτρους, οι οποίες κρατάνε τις προσωρινές τιμές.

Επανάληψη vs. Αναδρομή

Πλεονεκτήματα

- Η εκτέλεση της αναδρομής καταναλώνει λιγότερο χώρο στη μνήμη.
  - Βελτιστοποίηση ουραίας κλήσης (**tail-recursion optimization).**
  - Η αναδρομική κλήση είναι τελευταία στον ορισμό του κατηγορήματος, και η Prolog δε χρειάζεται να τοποθετήσει στη στοίβα στοιχεία για τις μεταβλητές καθώς και για τα σημεία επιστροφής και οπισθοδρόμησης.
- Πολλές φορές, προγράμματα που δεν "τρέχουν" λόγω μνήμης, με τέτοιες αλλαγές στον κώδικα είναι σε θέση να τρέξουν.

Επανάληψη vs. Αναδρομή

Μειονεκτήματα

- Ο κώδικας είναι λιγότερο "λογικός" ή "δηλωτικός" και περισσότερο "διαδικαστικός".
  - Δηλαδή η Prolog θυμίζει τις κλασικές γλώσσες προγραμματισμού!
- Χρειάζεται παραπάνω κατηγορήματα και ορίσματα.
- Στις διαδικασίες χειρισμού λιστών, υπάρχει περίπτωση οι λύσεις να επιστρέφονται αντεστραμμένες.
  - Χρειάζεται η χρήση του κατηγορήματος **reverse/2.**

list_max/2 – επαναληπτικός ορισμός

- Διαφέρει ως προς την αρχική τιμή.
- Η αρχική τιμή είναι το πρώτο στοιχείο της λίστας.
- Αν η λίστα είναι κενή, θα υπάρξει απευθείας αποτυχία.

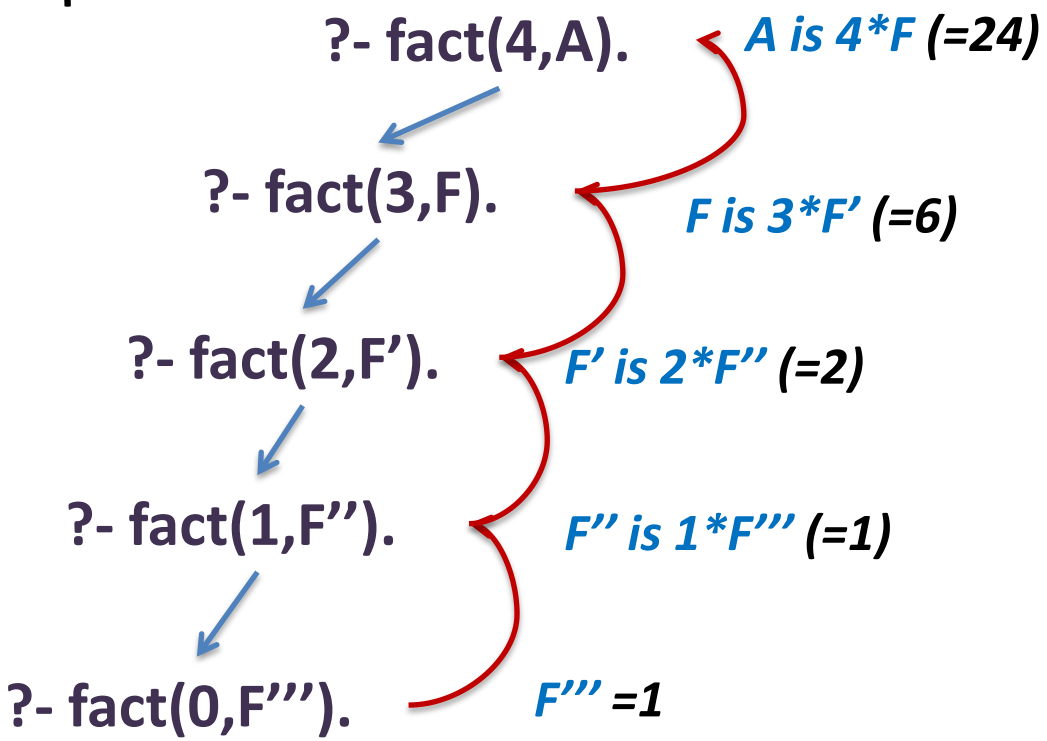
```
list_max([H|T],Max) :-
    list_max_aux(T,H,Max).

list_max_aux([],Max,Max).
list_max_aux([H|T],Temp,Max) :-
    max(H,Temp,Next),
    list_max_aux(T,Next,Max).
```

```
max(X,Y,X) :- X>Y.
max(X,Y,Y) :- X=<Y.
```

Αριθμητικές Διαδικασίες και Αναδρομή

Να ορισθεί κατηγόρημα για
τον υπολογισμό παραγοντικού

```
fact(0,1).
fact(N,F) :-
        N > 0,
        N1 is N – 1,
        fact(N1,F1),
        F is N * F1.
```



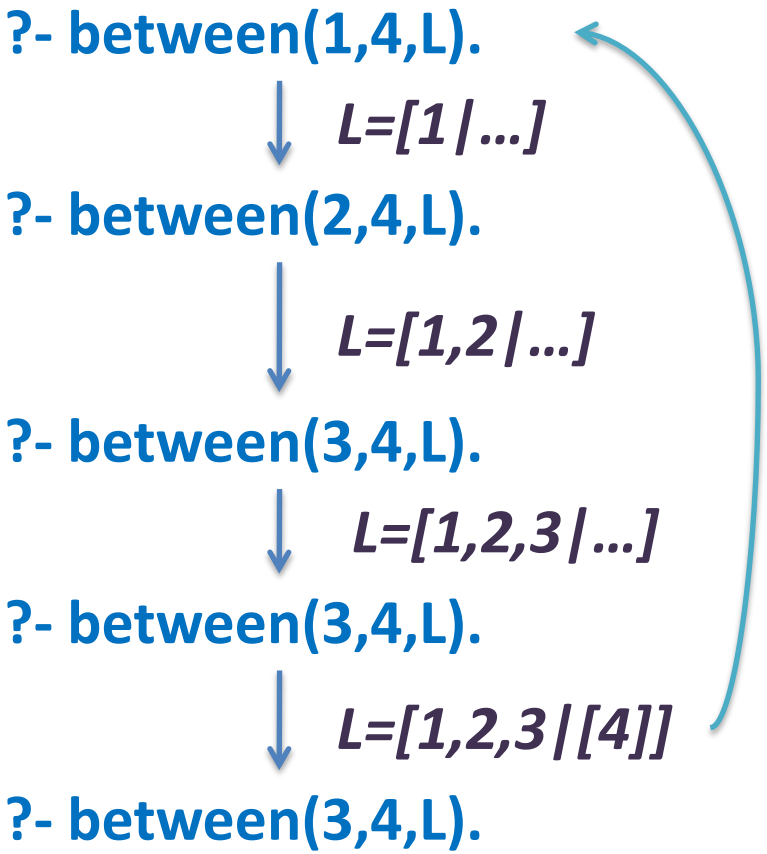
Αριθμητικές Διαδικασίες και Αναδρομή

- Να ορισθεί η διαδικασία **between(N1,N2,X**), η οποία να παράγει όλους τους ακεραίους στο διάστημα **[N1,N2].**

?- between(1,4,L).
L=[1,2,3,4]

```
between(N,N,[N]).
between(N1,N2,[N1|T]) :-
        N1 < N2,
        NewN1 is N1 + 1,
        between(NewN1,N2,T).
```

Βήματα εκτέλεσης και σχηματισμού λύσης



Αριθμητικές Διαδικασίες και Λίστες (1/3)

- Να ορισθεί η σχέση **length** που να επιστρέφει το μήκος μιας λίστας στοιχείων.
  - Μοιάζει με την **sum_list/2**, μόνο που προσθέτει **1** κάθε φορά αντί της κεφαλής της λίστας.

```
length([],0).
length([H|T],X) :-
        length(T,X1),
        X is X1 + 1.
```

```
?- length([a,b,c,d],A).
A=4
```

```
sum_list([],0).
sum_list([H|T], X) :-
        sum_list(T, X1),
        X is X1 + H.
```

Αριθμητικές Διαδικασίες και Λίστες (2/3)

- Να ορισθεί διαδικασία για το εσωτερικό γινόμενο 2 διανυσμάτων.
  - Το διάνυσμα ν-διαστάσεων αναπαρίσταται με λίστα ν-στοιχείων.
  - Οι λίστες-διανύσματα θεωρούνται ισομήκεις.

```
list_prod([],[],0).
list_prod([X1|T1],[X2|T2], P) :-
        list_prod(T1, T2, P2),
        P is X1*X2+ P2.
```

Αριθμητικές Διαδικασίες και Λίστες (3/3)

- Να γραφεί πρόγραμμα που να υπολογίζει το διανυσματικό άθροισμα 2 διανυσμάτων.

```
addlist([],[],[]).
addlist([X|T1],[Y|T2],[Z|T3]) :-
        Z is X + Y,
        addlist(T1,T2,T3).
```

Ν-οστο μέλος λίστας

- Κατηγόρημα που να επιστρέφει το ν-οστό μέλος λίστας.

```
nth_member(1,E,[E|_]).
nth_member(N,E,[H|T]) :-
    N > 1,
    N1 is N - 1,
    nth_member(N1,E,T).
```

Ορθή χρήση

- Παράδειγμα:

```
?- nth_member(3,A,[a,b,c,d,e,c]).
A = c
?- nth_member(3,c,[a,b,c,d,e,c]).
yes
?- nth_member(3,b,[a,b,c,d,e,c]).
no
?- nth_member(7,g,[a,b,c,d,e,c]).
no
```

Λάθος Χρήση

- Βρες τη θέση κάποιου στοιχείου στη λίστα:

```
?- nth_member(X,c,[a,b,c,d,e]).
! ----------------------------
! Error 22 : Instantiation Error
! Goal     : _309496 is _301876

Aborted
```

Λάθος Χρήση - Εξήγηση

```
nth_member(N,E,[H|T]) :-
    N > 1,
    N1 is N - 1,
    nth_member(N1,E,T).
```

- Οι ελεύθερες μεταβλητές δεν μπορούν να συμμετέχουν σε συγκρίσεις ούτε σε αριθμητικές εκφράσεις.

Εύρεση θέσης στοιχείου

- Κατηγόρημα που να επιστρέφει τη θέση (με αριθμό) ενός στοιχείου σε μια λίστα.

```
nth_member(1,E,[E|_]).
nth_member(N,E,[H|T]) :-
    nth_member(N1,E,T),
    N is N1 + 1.
```

Αν το βρω στην κεφαλή, τότε είναι στη θέση 1

Αν όχι, τότε το ψάχνω στη ουρά. Αν το βρω στην ουρά στη θέση N1, τότε βρίσκεται στη θέση N=N1+1 της αρχικής λίστας

Σωστή Χρήση

```
?- nth_member(X,c,[a,b,c,d,e,c]).
X = 3 ;
X = 6 ;
no
?- nth_member(X,g,[a,b,c,d,e,c]).
no
?- nth_member(3,c,[a,b,c,d,e,c]).
yes
```

Χρήση για εύρεση ν-οστού στοιχείου

?- nth_member(3,A,[a,b,c,d,e,c]).

A = c

- Φαίνεται να καλύπτει τις απαιτήσεις και των 2 τρόπων χρήσης!

Αλλά!! ***10000 στοιχεία!***

?- nth_member(10000,A,[a,...,c]).

Error 2, Local Stack Full, Trying is/2

Aborted

Μη-αποδοτικός ορισμός

- Ενώ ξέρω εκ των προτέρων πόσες φορές το πολύ θα επαναληφθεί η αναδρομή, δεν το εκμεταλλεύομαι.
- Ουσιαστικά η Prolog δοκιμάζει όλα τα στοιχεία της λίστας μέχρι η θέση του στοιχείου να συμπέσει με τον αριθμό Ν που έδωσε ο χρήστης.
- Είναι πολύ χρονοβόρα και μνημοβόρα διαδικασία!

Συνδυασμός των 2 ορισμών

```
nth_member(1,E,[E|_]).
nth_member(N,E,[H|T]) :-
    nonvar(N),
    N > 1,
    N1 is N - 1,
    nth_member(N1,E,T).
nth_member(N,E,[H|T]) :-
    var(N),
    nth_member(N1,E,T),
    N is N1 + 1.
```

- Ελέγχει αν το N είναι σταθερά at-run-time
- Χρησιμοποιείται όταν δίνεται η θέση του στοιχείου της λίστας

- Ελέγχει αν το N είναι μεταβλητή at-run-time
- Χρησιμοποιείται όταν ζητείται η θέση του στοιχείου της λίστας

Διαδικασίες Εισόδου-Εξόδου

- Στην Prolog, η επικοινωνία μεταξύ χρήστη και προγράμ-ματος γίνεται συνήθως με τη μορφή ερωτήσεων από την πλευρά του χρήστη και απαντήσεων από την πλευρά του συστήματος.
  - Σε πολλές περιπτώσεις απαιτείται μεγαλύτερη διαλογικότητα μεταξύ χρήστη και προγράμματος.
- Αυτό μπορεί να επιτευχθεί με ενσωματωμένα κατηγορήματα τα οποία
  - διαβάζουν όρους ή χαρακτήρες από το προκαθορισμένο κανάλι εισόδου (πληκτρολόγιο ή αρχείο) και
  - επιστρέφουν το αποτέλεσμα στο προκαθορισμένο κανάλι εξόδου (οθόνη ή αρχείο).

Είσοδος Όρων από το Πληκτρολόγιο

- Υπάρχει το κατηγόρημα **read(X)**, το οποίο διαβάζει τον επόμενο **όρο** που εισάγεται από το πληκτρολόγιο.

*Ο συνδυασμός χαρακτήρων **|:** εμφανίζεται από το σύστημα για να υποδηλώσει ότι περιμένει την πληκτρολόγηση κάποιο όρου.*

?- read(X).

- *Το άτομο **prolog** και η τελεία "**.**" πληκτρολογούνται από τον χρήστη.*
- *Η τελεία είναι απαραίτητη για να δηλώσει το τέλος του όρου.*

|: *prolog.*

X = prolog

*Η μεταβλητή **X** που υπάρχει στην αρχική κλήση ενοποιείται με τον όρο που πληκτρολογήθηκε.*

Είσοδος Χαρακτήρων

- Κατηγόρημα **get(X)**, το οποίο διαβάζει έναν χαρακτήρα και επιστρέφει τον ASCII κωδικό του.
  - Επιστρέφει μόνο «εκτυπώσιμους» χαρακτήρες (όχι space, κλπ).

?- get(X).

|: a

X = 97

Πρέπει να πατηθεί enter.

- Κατηγόρημα **get0(X)**, το οποίο διαβάζει έναν οποιονδήποτε χαρακτήρα και επιστρέφει τον ASCII κωδικό του.

?- get0(X).

|: (space)

X = 32

Έξοδος Όρων στην Οθόνη

- Χρησιμοποιείται το κατηγόρημα **write(X)**, το οποίο τυπώνει τον όρο Χ στην οθόνη.

?- write(prolog), write(lisp).

prologlispyes

- Για να αλλάξει η γραμμή εκτύπωσης χρησιμοποιείται το κατηγόρημα **nl.**

?- write(prolog), nl, write(lisp), nl.

prolog

lisp

yes

Έξοδος Χαρακτήρων

- **put(X)** - εκτυπώνει έναν χαρακτήρα στην οθόνη.
  - Χ είναι ο ASCII κωδικός του.

?- put(65).

A

- Εκτυπώνει και μη «εκτυπώσιμους» χαρακτήρες.

?- put(65), put(32), put(65).

A A

- Αν δεν ξέρουμε τον ASCII κωδικό μπορούμε να χρησιμοποιήσουμε τον χαρακτήρα με quotes.

?- put('A').

A

?- put("A").

A

Ασκήσεις - Είσοδος/Εξοδος (1/3)

Να γραφεί πρόγραμμα το οποίο να τυπώνει τα στοιχεία μιας λίστας σε διαφορετικές σειρές.

```
writelist([]).
writelist([X|L]) :-
    write(X), nl,
    writelist(L).
```

```
?- writelist([hello,prolog,world]).
hello
prolog
world
yes
```

Ασκήσεις - Είσοδος/Εξοδος (2/3)

- Να γραφεί πρόγραμμα που να τυπώνει τις υπολίστες μιας λίστας σε διαφορετικές σειρές.

```
writelist2([]).
writelist2([X|T]) :-
    writeline(X), nl,
    writelist2(T).
writeline([]).
writeline([X|T]) :-
    write(X), put(32),
    writeline(T).
```

Τυπώνει τα στοιχεία μιας λίστας σε μια σειρά.

```
?- writelist2([[hello,prolog,world],
               [how,are,you]]).
hello prolog world
how are you
yes
```

Είσοδος/Εξοδος Όρων σε Αρχεία

- Χρησιμοποιούνται τα ίδια κατηγορήματα αφού ανακατευθυνθούν πρώτα τα κανάλια εισόδου/εξόδου σε αρχεία ή/και συσκευές (π.χ. εκτυπωτής).
- **see(*Όνομα_Αρχείου*)**: Ορίζει το κανάλι εισόδου.
  - Μετά την εκτέλεση του κατηγορήματος αυτού, όλες οι εντολές εισόδου αναφέρονται στο ***Όνομα_Αρχείου*** (κανάλι).
- **seen**: Ακυρώνει το κανάλι επικοινωνίας που ορίστηκε με την διαδικασία **see** και κλείνει όλα τα αρχεία εισόδου.
- **tell(*Όνομα_Αρχείου*)**: Ορίζει το κανάλι εξόδου.
  - Μετά την εκτέλεση του κατηγορήματος αυτού, όλες οι εντολές εξόδου αναφέρονται στο ***Όνομα_Αρχείου***.
- **told**: Ακυρώνει το κανάλι επικοινωνίας που ορίστηκε με την διαδικασία **tell** και κλείνει όλα τα αρχεία εξόδου.

Παράδειγμα Εξόδου σε Αρχείο

- Το κατηγόρημα **out/2** δέχεται σαν είσοδο μια λίστα και το όνομα ενός αρχείου.
  - Γράφει τα στοιχεία της λίστας στο αρχείο.
  1. Το **tell** ανακατευθύνει την έξοδο στο αρχείο.
  2. Το **writelist** τυπώνει τα στοιχεία της λίστας.
  3. Το **told** καθιστά και πάλι κανάλι εξόδου την οθόνη.

```
out(L,F):-
    tell(F),
    writelist(L),
    told.
```

```
?- out([hello,prolog,world],'myfile.txt').
yes
```

Παράδειγμα (βιβλίου)

- Να γραφεί διαδικασία που να γράφει τα στοιχεία μιας λίστας κυκλικά στα αρχεία **test1, test2** και στην οθόνη.

```
out(X) :- send(X,test1).
send([],_) :- told.
send([X|T],F) :-
    tell(F),
    write(X),
    next_out(F,F1),
    send(T,F1).
```

Κυκλική Εναλλαγή

```
next_out(test1,test2).
next_out(test2,user).
next_out(user,test1).
```

?- out([1,2,3,4,5,6]).

- Θα δούμε στην οθόνη το **3** και το **6** και τα υπόλοιπα στα αρχεία **test1** και **test2** (ASCII files).

Ασκήσεις - Είσοδος/Εξοδος (3/3)

- Να γραφεί πρόγραμμα που να διαβάζει ένα κείμενο από ένα αρχείο και να το γράφει σ' ένα άλλο αρχείο.

```
file_copy(F1, F2) :-
    see(F1), tell(F2),
    copytext.
```

```
?- file_copy(test1,test2).
```

- *Πρόβλημα: αγνοεί τα κενά.*

```
copytext :-
    get(Ch),
    copy1(Ch).
```

```
copy1(46) :-
    put(46), seen, told.
copy1(Ch) :-
    put(Ch),
    copytext.
```

Τερματισμός

Επανάληψη

Χειρισμός Συμβολοσειρών (1/2)

- Στην Prolog, η τακτική που ακολουθείται για τον χειρισμό των συμβολοσειρών (strings) είναι η μετατροπή τους σε λίστες και η χρήση κατηγορημάτων επεξεργασίας λιστών.
- Η Prolog θεωρεί πως οποιαδήποτε συμβολοσειρά βρίσκεται μέσα σε διπλά εισαγωγικά ισοδυναμεί με λίστα ASCII κωδικών.
  - Οπουδήποτε απαιτείται λίστα ASCII κωδικών μπορούμε να βάζουμε συμβολοσειρά σε διπλά εισαγωγικά.

?- X="prolog".

X = [112,114,111,108,111,103]

Χειρισμός Συμβολοσειρών (2/2)

- Υπάρχει το κατηγόρημα **name**, το οποίο μετατρέπει ένα άτομο σε λίστα ASCII κωδικών και αντίστροφα:

?- name(prolog,X).

X = [112,114,111,108,111,103]

?- name(X,[112,114,111,108,111,103]).

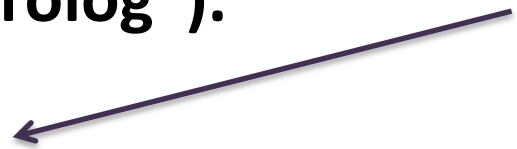
X = prolog

?- name(X,"prolog").

X = prolog

?- name(‘Prolog’,“Prolog”).

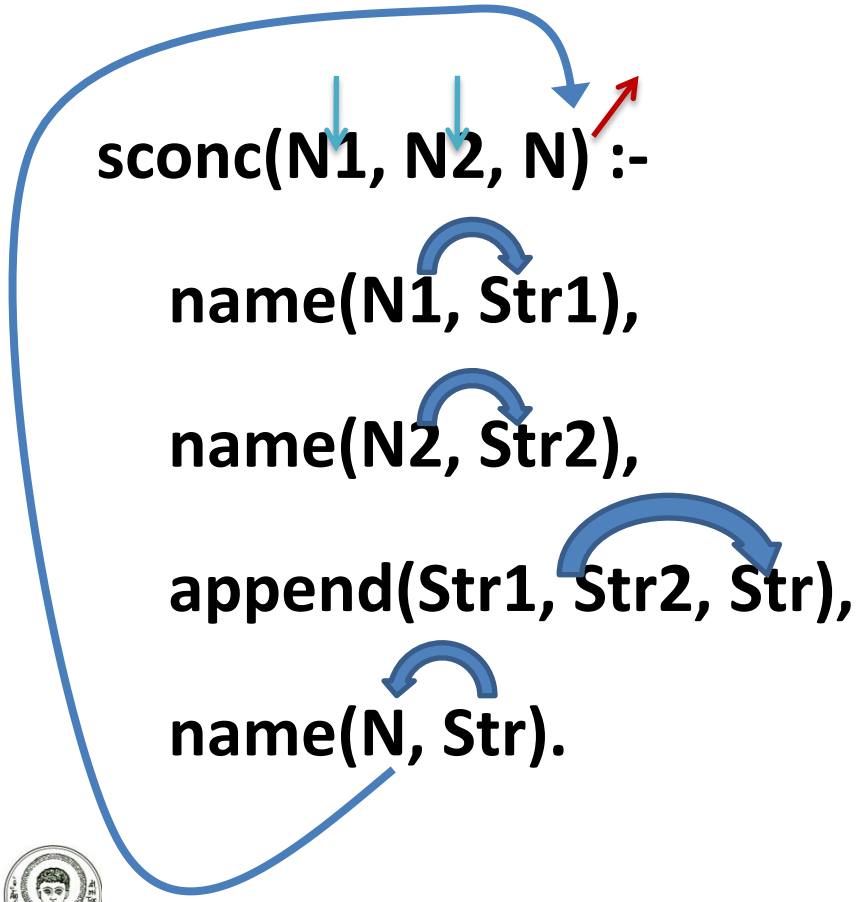
yes



Επειδή το άτομο ξεκινάει από κεφαλαίο χαρακτήρα πρέπει να μπει σε απλά quotes.

Παράδειγμα με name/2

- Συνένωση 2 strings.



?- sconc(prolog, isgood, X).

X= prologisgood

- *Δεν χρειάζονται εισαγωγικά.*

Ασκήσεις Χειρισμού Συμβολοσειρών (1/4)

- Να ορισθεί η σχέση **starts(Atom,Char)**, η οποία θα ελέγχει αν το **Atom** αρχίζει με τον χαρακτήρα **Char**.

```
starts(Atom,Char) :-
        name(Atom,[X|T]),
        name(Char,[X]).
```

?- starts(star,s).

yes

?- starts(star,A).

A = s

Ασκήσεις Χειρισμού Συμβολοσειρών (2a/4)

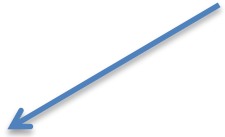
- Να ορισθεί η σχέση **plural** που να μετατρέπει ονόματα στον πληθυντικό.

?- plural (table, X).

X= tables

```
plural(N,Ns) :-
        name(N,List),
        name(s,CodeS),
        append(List,CodeS,NewList),
        name(Ns,NewList).
```

*Μετατρέπει το χαρακτήρα **s** σε λίστα με έναν ASCII κωδικό.*



Ασκήσεις Χειρισμού Συμβολοσειρών (2b/4)

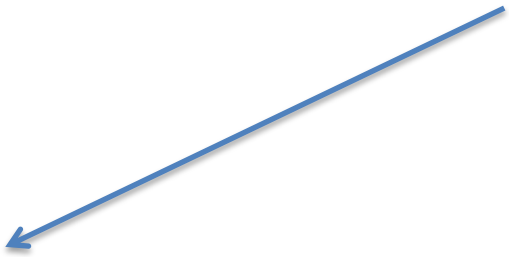
- Να ορισθεί η σχέση **plural** που να μετατρέπει ονόματα στον πληθυντικό.

?- plural (table, X).

X= tables

```
plural(N,Ns) :-
        name(N,List),
        name(s,CodeS),
        append(List,CodeS,NewList),
        name(Ns,NewList).
```

Εναλλακτικά



Ασκήσεις Χειρισμού Συμβολοσειρών (2c/4)

- Να ορισθεί η σχέση **plural** που να μετατρέπει ονόματα στον πληθυντικό.

?- plural (table, X).

X= tables

Εναλλακτικά

```
plural(N,Ns) :-
        name(N,List),
        append(List,"s",NewList),
        name(Ns,NewList).
```

Ασκήσεις Χειρισμού Συμβολοσειρών (3a/4)

- Να ορισθεί μια διαδικασία που να ελέγχει αν ένα string είναι παλινδρομικό.

```
palindstring(S) :-
    name(S,NS),
    palindrome(NS).
```

```
?- palindstring(anna).
yes
```

- **palindrome/1** είναι το κατηγόρημα που ελέγχει αν μία λίστα είναι παλινδρομική:

```
palindrome(L) :- reverse(L,L).
```

Ασκήσεις Χειρισμού Συμβολοσειρών (3b/4)

- *Άλλη λύση:*

```
palindstring(S) :-
        revstring(S,S).

revstring(S,RS) :-
        name(S,L),
        reverse(L,RL),
        name(RS,RL).
```

Ασκήσεις Χειρισμού Συμβολοσειρών (4/4)

- Να γραφεί κατηγόρημα που διαγράφει από μια συμβολοσειρά κάποια συγκεκριμένη ακολουθία χαρακτήρων, όσες φορές εμφανίζεται.

```
?- remove(cd, abcdecdx, X).
X = abex

remove(S1,S,NewS) :-
    name(S1,S1L),
    name(S,SL),
    del_all(S1L,SL,NewSL),
    name(NewS,NewSL).

del_all(X,L,NewL) :-
    delstring(X,L,SubL),
    del_all(X,SubL,NewL).
del_all(X,L,L).    % έξοδος

delstring(X,L,NewL) :-
    append(L1,L2,L),    % χωρίζει την L σε L1 και L2
    append(X,L3,L2),    % χωρίζει την L2 σε X και L3
    append(L1,L3,NewL). % συνδέει την L1 και L3 (σβήνει το X)
```

επανάληψη

Υπάρχει περίπτωση να αποτύχει

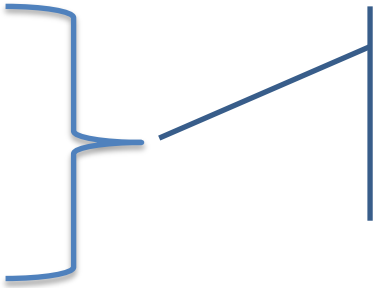
Διόρθωση ορθογραφίας λέξεων

- Έστω ότι έχουμε μία βάση δεδομένων (γεγονότων) με «λέξεις», οι οποίες είναι γραμμένες «ορθά».
  - Π.χ.

word([h,e,l,l,o]).

word([m,a,n]).

word([w,o,m,a,n]).



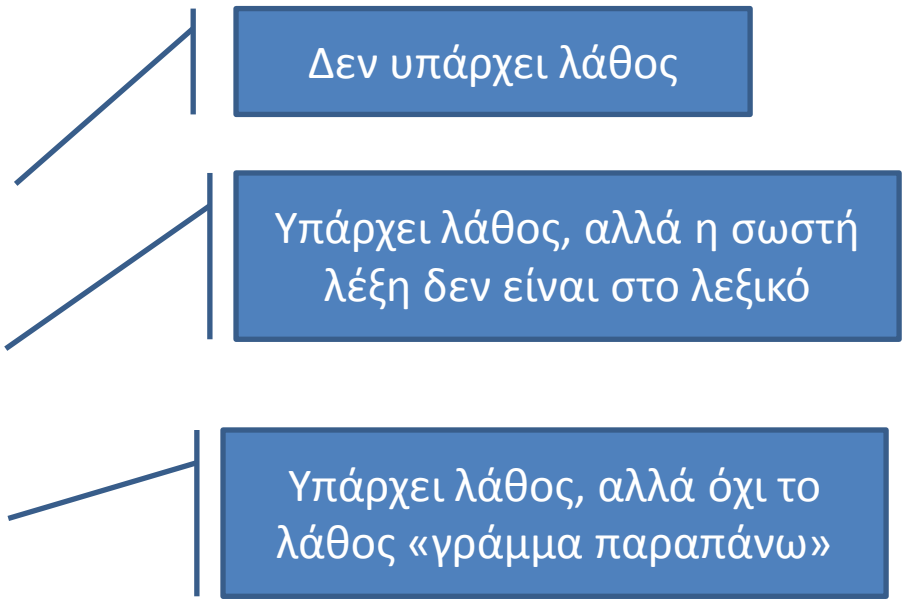
Θεωρούμε ότι κάθε λέξη είναι μια λίστα από γράμματα

- Να φτιαχτεί κατηγόρημα που να δέχεται μια λέξη με το ορθογραφικό λάθος «γράμμα παραπάνω» και να το διορθώνει επιστρέφοντας την σωστή λέξη από το λεξικό.

Ορθογραφία: Γράμμα παραπάνω

```
extra_letter(Wrong,Correct) :-
        word(Correct),
        append(A,B,Correct),
        append(A,[X|B],Wrong).
```

```
?- extra_letter([h,e,l,l,o,w],A).
A = [h,e,l,l,o]
?- extra_letter([w,o,m,a,n],A).
no
?- extra_letter([h,o,u,m,e],A).
No
?- extra_letter([w,u,m,a,n],A).
no
```



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.