

### Χρήσεις του assert/retract

- Προσθήκη στο πρόγραμμα της λύσης ενός προβλήματος:

```
?- solve(Problem,Answer), asserta(solve(Problem,Answer)).
```

↑  
χωρίς τιμή
↑  
Δεν εκτελείται πάλι
↑  
με τιμή

- Αν υποβληθεί στο πρόγραμμα το ίδιο “**Problem**” δεν θα χρειαστεί να ξανα-υπολογιστεί το “**Answer**”, γιατί θα υπάρχει σαν γεγονός (fact).
  - Ονομάζεται caching.

45

### Παράδειγμα: Παραγοντικό

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

46

### Χρήσεις του assert/retract

- Προσομοίωση Καθολικών Μεταβλητών (global variables)
  - Αποθηκεύουμε την τιμή μιας μεταβλητής για να χρησιμοποιηθεί από όλες τις προτάσεις του προγράμματος.
  - Παράδειγμα:** Έστω ότι έχουμε τα εξής γεγονότα:

```
next('Greece','Turkey'). next('Greece','Albania'). next('Greece','Bulgaria').
next('Greece','FYROM'). next('Turkey','Iraq'). ...
```

και θέλουμε ένα πρόγραμμα το οποίο θα βρίσκει με πόσες και ποιες χώρες συνορεύει μια χώρα.

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

```
?- count('Greece',N).
Turkey
Albania
Bulgaria
FYROM
4
```

46

### Ασκήσεις assert/retract

- Αν μια βάση δεδομένων περιέχει προτάσεις της μορφής :

```
product(0,0,0).    product(1,2,2).    product(0,1,0).    product(1,1,1).
```

- Να γραφεί ερώτηση που τις διαγράφει όλες
- Να γραφεί ερώτηση που διαγράφει αυτές που παράγουν 0.

#### Λύση:

- ?- retract(product(X,Y,Z)), fail.
- ?- retract(product(X,Y,0)), fail.

47

### Ασκήσεις assert/retract

- Να ορισθεί η σχέση **copy (Term, Copy)** έτσι ώστε το **Copy** να είναι αντίγραφο του **Term** με διαφορετικά ονόματα μεταβλητών.

- Χρησιμοποιώντας **assert, retract**
- Χρησιμοποιώντας **bagof**

#### Λύση:

α) **copy (Term, Copy) :- asserta (term (Term) ), retract (term (Copy)) .**

Ταυτόχρονα δίνει τιμή στη μεταβλητή **Copy = Term**

β) **copy (Term, Copy) :- bagof (X, X = Term, [Copy]) .**

**Εξήγηση:** Η **Copy** θα πάρει σαν τιμή το μοναδικό στοιχείο της λίστας που θα είναι η τιμή του **X** που θα είναι ίση με **Term**.

48

### Διαχείριση Γεγονότων

- ❖ Να γραφεί πρόγραμμα (με μενού επιλογών) διαχείρισης γεγονότων για βαθμούς φοιτητών.
- ❖ Βασικό μενού επιλογών:

```
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).
```

50

### Διαχείριση Γεγονότων

- ❖ "Φόρτωμα" γεγονότων από αρχείο:

```
do_on_choice(1) :-
    !, 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

ins_fact(end_of_file) :- !.      % Τέλος αρχείου
ins_fact(Fact) :- assert(Fact).  % Εισαγωγή γεγονότος
```

51

### Διαχείριση Γεγονότων

- ❖ 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

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
```

52

## Διαχείριση Γεγονότων

❖ Αναζήτηση γεγονότος:

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

find_student(Name) :-          % Αν ο φοιτητής υπάρχει, τότε τύπωσε το βαθμό του
    student(Name,Grade), !,
    write('Grade: '),
    write(Grade), nl.

find_student(_) :-            % Αν όχι, τύπωσε ότι δε βρέθηκε
    write('Not found!'), nl.
```

53

## Διαχείριση Γεγονότων

❖ 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,
    assert(student(Name,Grade)), % Εισαγωγή γεγονότος
    nl,
    run.
```

54

## Διαχείριση Γεγονότων

❖ 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

% Αν ο φοιτητής υπάρχει, τότε διέγραψε το γεγονός (βαθμός αδιάφορος)

```
delete_student(Name) :-
    retract(student(Name,_)), !.
```

% Αν όχι, τύπωσε ότι δε βρέθηκε

```
delete_student(_) :-
    write('Not found!'), nl.
```

55

## Διαχείριση Γεγονότων

❖ 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),
    average_list(L,Avg),      % Βρίσκει το μέσο όρο των αριθμών της λίστας
    nl, write('Course Average: '), write(Avg), nl, nl,
    run.
```

- Πρέπει να χρησιμοποιηθεί η `sum_list/2` για να αθροίσει τα στοιχεία της λίστας και η `length/2` για να βρεθεί το πλήθος των στοιχείων.
- Στη συνέχεια αρκεί μια διαίρεση.

56

## Διαχείριση Γεγονότων

❖ 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 (γι' αυτό μπαίνει πρώτος)  
    setof(G-S,student(S,G),L) ,  
    print_students(L) ,                % Τυπώνει τη λίστα  
    nl, run.
```

*% Δεν υπάρχουν άλλοι φοιτητές - τύπωσε κενή γραμμή και σταμάτα*

```
print_students([]) :- nl.
```

*% Τύπωσε τον πρώτο φοιτητή και τον βαθμό του και μετά αναδρομικά τους υπόλοιπους*

```
print_students([G-S|T]) :-  
    write(S), write(' : '), write(G), nl,  
    print_students(T).
```

## Διαχείριση Γεγονότων

❖ Έξοδος:

```
do_on_choice(0) :- !,    % Η αποκοπή "κόβει" τις άλλες επιλογές  
    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 5!'), nl,  
    run.
```