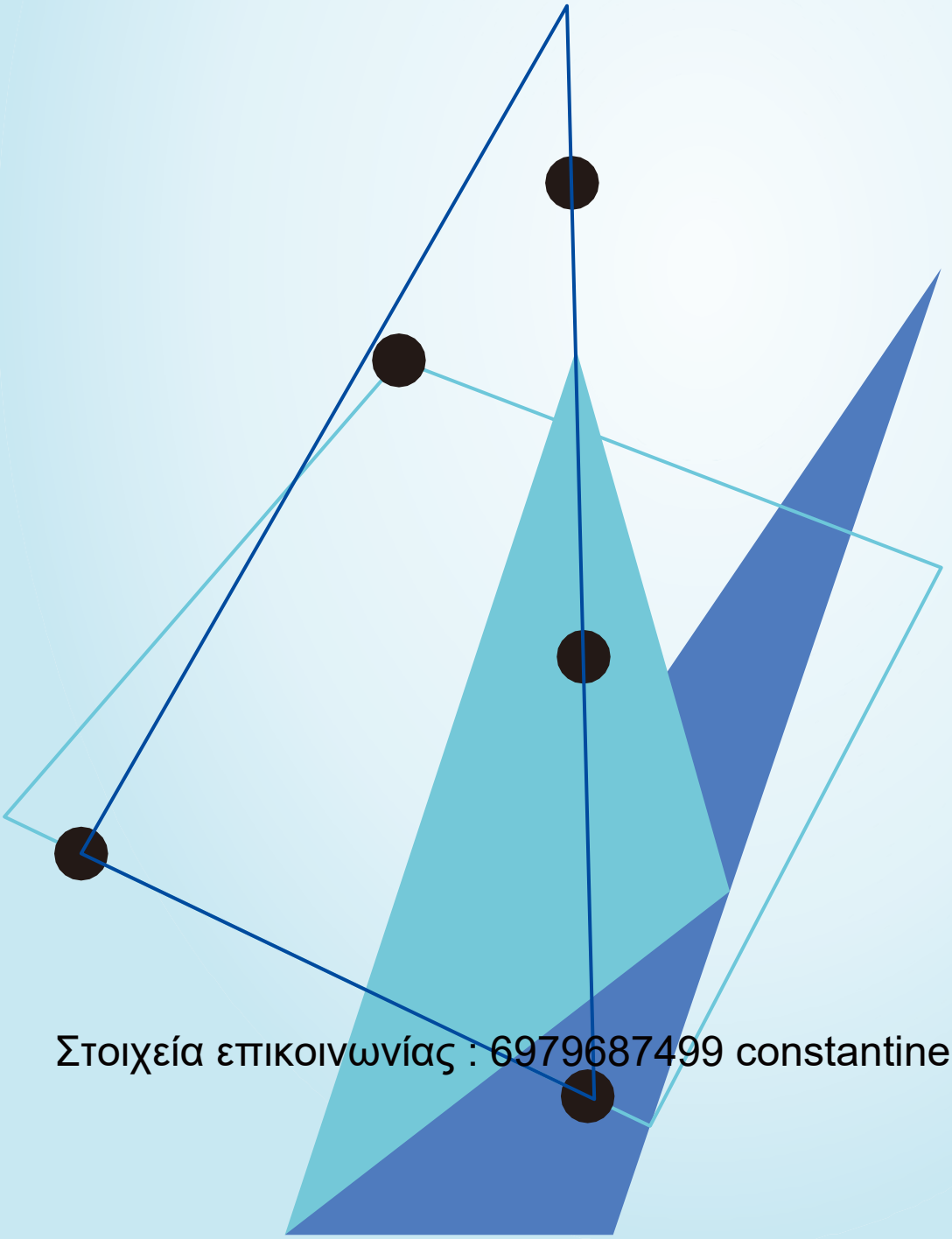
Τμήμα Πληροφορικής Πανεπιστήμιο Πειραιά

Πρόγραμμα Μεταπτυχιακών Σπουδών

# Εργασία Λογικός Προγραμματισμός Μέρος Α

ΧΡΟΝΟΠΟΥΛΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

AM: MPPL 21081



Στοιχεία επικοινωνίας : 6979687499 constantine1212204@gmail.com

# ΕΙΣΑΓΩΓΗ

Γλώσσα προγραμματισμού Prolog

Η Prolog είναι μια γλώσσα λογικού προγραμματισμού που σχετίζεται με την τεχνίτη νοημοσύνη και την υπολογιστική γλωσσολογία (computational linguistics) .
Αντίθετα με πολλές άλλες γλώσσες προγραμματισμού η Prolog intended κύριος ως δηλωτική γλώσσα προγραμματισμού δηλαδή τα προγράμματα εκφράζονται ως σχέσεις ,εκφρασμένες ως γεγονότα και κανόνες .

Η γλώσσα αναπτύχθηκε το 1972 στην Μασσαλία της Γαλλίας από τους Alain Colmerauer και Robert Kowalski. Η Prolog είναι μια από της πρώτες και παραμένει η πιο δημοφιλής τέτοια γλώσσα με πολλές δωρεάν αλλά και κερδοσκοπικές εφαρμογές .Η γλώσσα έχει χρησιμοποιηθεί για αποδείξεις θεωρημάτων , εμπειρικά συστήματα , συστήματα τύπου και αυτοματοποιημένο σχεδιασμό .

Η Prolog είναι κατάλληλη για συγκεκριμένες εργασίες που επωφελούνται από λογικά ερωτήματα που βασίζονται σε κανόνες όπως αναζήτηση βάσης δεδομένων .

Λογικός Προγραμματισμός

Ο Λογικός Προγραμματισμός είναι ένας από τους κλάδους του προγραμματισμού υπολογιστών ,στον οποίο οι δηλώσεις του προγράμματος εκφράζουν τα γεγονότα και τους κανόνες για διαφορετικά προβλήματα μέσα σε ένα σύστημα τυπικής λογικής .

Σύνταξη και σημασιολογία

Στην Prolog, τα πρόγραμμα εκφράζεται με ορούς σχέσεων και ένας υπολογισμός ξεκινά εκτελώντας ένα ερώτημα πάνω σε αυτές τοις σχεσεις.Οι σχέσεις και τα ερωτήματα κατασκεπάζονται χρησιμοποιώντας τον μοναδικό τύπο δεδομένων της prolog τον ορό.

Τύποι Δεδομένων

Ο μοναδικός τύπος δεδομένων της Prolog είναι ο ορός .Οι οροί είναι άτομα , αριθμοί , μεταβλητές ή σύνθετοι οροί.

- Τα άτομα είναι γενικής χρυσής ονόματα χωρίς κάποιο συγκεκριμένο νόημα .Παραδείγματα ατόμων είναι τα : “x”, “red” , “some atom”.
- Οι αριθμοί μπορούν να είναι floats ή integers .
- Οι μεταβλητές συμβολίζονται από ένα string αποτελούμενο από γράμματα ,αριθμούς και την κάτω παύλα ”_” τα οποία όμως πρέπει να ξεκινάνε από κεφάλαιο γράμμα η “_” .
- Ένας σύνθετος ορός αποτελείται από ένα άτομο και έναν αριθμό επιχειρημάτων που είναι πάλι όροι . Οι σύνθετοι όροι γράφονται ως συντελεστής ακολουθούμενος από μια λίστα ορών διαχωρισμένη από κόμματα Πχ friends(maria,[tom,jim]) .

Ειδικές Περιπτώσεις

- Μια λίστα είναι μια διαταγμένη ομάδα ορών ,συμβολίζεται από τα square brackets με τους ορούς διαχωρισμένους με κόμματα πχ , [1,2,3] .

- Strings είναι μια σειρά χαρακτήρων μέσα σε εισαγωγικά πχ “Prolog is a logic programming language”.

Κανόνες και Γεγονότα

Τα προγράμματα στην Prolog περιγράφουν σχέσεις που ορίζονται με ρήτρες(clauses).Υπάρχουν δυο τυποι ρητρων γεγονότα και κανόνες .
Ένας κανόνας είναι της μορφής :

```
Head :- Body .
```

και διαβάζεται το head είναι αληθές εάν το Body είναι αληθές.
Ρητές με κενό σώμα λέγοντα Γεγονότα(fucts) πχ

```
cat(oscar).
```

Γιάννα ελέγξουμε εάν ο oscar είναι cat θα γράφαμε :

```
?- cat(oscar).
Yes
```

Η για να ρωτήσουμε πια πράγματα είναι cat θα γράφαμε:

```
?- cat(X).
X = oscar
```

Αν προσθέταμε τον κανόνα :

```
animal(X) :- cat(X).
```

και ρωτάγαμε :

```
?-animal(X).
X = oscar
```

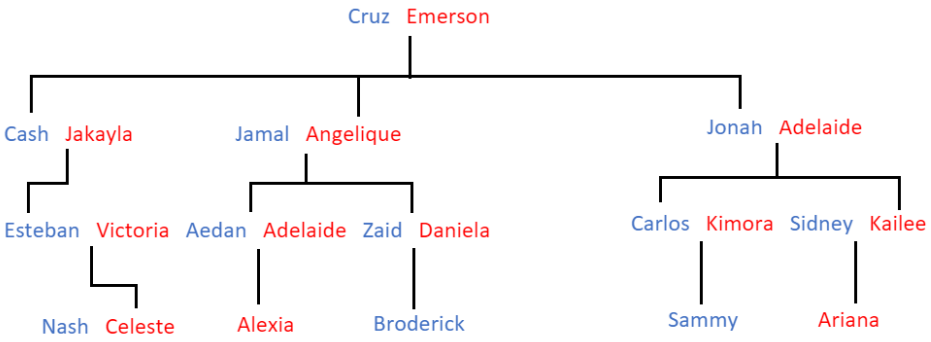
# Θέμα 1

Θα χρησιμοποιήσουμε τα κατηγορήματα :

parent(Parent,Child).
male(Name).
female(Name).

Δηλ. το Parent ειναι γονέας του Child.

Θα χρησιμοποιήσουμε ως παράδειγμα  το γενεαλογικό δέντρο του διαγράμματος :




Το οποίο θα το εκφράσουμε στην prolog με τα facts :

```
/*-------------------------------------*/
/* facts                               */
/*-------------------------------------*/
male(cruz).
male(cash).
male(jamal).
male(esteban).
male(aedan).
male(jonah).
male(carlos).
male(sidney).
male(zaid).
male(sammy).
male(nash).
male(broderick).
female(emerson).
```

```
female(jakayla).
female(angelique).
female(victoria).
female(adelaide).
female(kimora).
female(kailee).
female(daniela).
female(ariana).
female(celeste).
female(alexia).
female(francesca).


parent(esteban,celeste).
parent(victoria,celeste).
parent(cash,esteban).
parent(jakayla,esteban).
parent(cruz,cash).
parent(emerson,cash).
parent(aedan,alexia).
parent(adelaide,alexia).
parent(jamal,aedan).
parent(angelique,aedan).
parent(cruz,angelique).
parent(emerson,angelique).
parent(carlos,sammy).
parent(kimora,sammy).
parent(sidney,ariana).
parent(kailee,ariana).
parent(jonah,carlos).
parent(adelaide,carlos).
parent(jonah,kailee).
parent(adelaide,kailee).
parent(cruz,jonah).
parent(emerson,jonah).
```

## Κανόνας 1

sibling(X,Y) :- parent(A,X),parent(A,Y), X\=Y.

Οι X και Y είναι αδέλφια αν υπάρχει A που να είναι γονέας του X και Γονέας του Y οπου X και Y είναι διαφορετικά .

```
?- sibling(cash,angelique).
true .

?- sibling(cash,X).
X = angelique .

?- sibling(cruz,emerson).
false.
```

## Κανόνας 2

father(X,Y) :- male(X),parent(X,Y).

Ο X είναι πατέρας του Y αν ο X είναι άντρας και γονέας του Y .

```
?- father(cruz,cash).
false.

?- father(cash,cruz).
true .
```

## Κανόνας 3

mother(X,Y) :- female(X),parent(X,Y).

Ο Χ ειναι μητέρα του Υ αν ο Χ είναι Γυναικά και γονέας του Υ

```
?- mother(aedan,angelique).
true.

?- mother(aedan,emerson).
false.
```

## Κανόνας 4

grandfather(X,Y) :- male(X),parent(X,S),parent(S,Y).

Ο Χ είναι πάππους του Υ αν ο Χ είναι άντρας και είναι γονέας του γονέα του Υ .

```
?- grandfather(esteban,cruz).
true .

?- grandfather(esteban,emerson).
false.
```

## Κανόνας 5

grandmother(X,Y) :- female(X),parent(X,S),parent(S,Y).

Ο Χ είναι Γιαγιά του Υ αν ο Χ είναι γυναικά και είναι γονέας του γονέα του Υ .

```
?- grandmother(esteban,emerson).
true
Unknown action: [] (h for help)
Action? .

?- grandmother(esteban,cruz).
false.
```

## Κανόνας 6

allkids(X,List) :- findall(Y,parent(X,Y),List).

Βάζουμε στην λίστα List όλα τα Y που έχουν parent τον X

```
?- allkids(cruz,X).
X = [cash, angelique, jonah].
```

## Κανόνας 7

cousin(X,Y) :- grandfather(G,X),grandfather(G,Y);
grandmother(G,X),grandmother(G,Y),X\=Y.

Οι X και Y είναι ξαδέλφια αν ο έχουν τον ίδιο παππού η γιαγιά.

```
?- cousin(esteban,X).
X = esteban ;
X = aedan ;
X = carlos ;
X = kailee .
```

## Κανόνας 8

seccousin(X,Y) :- parent(P1,X),cousin(P1,P2), parent(P2,X),X\=Y.

Οι X και Y είναι ξαδέλφια αν ο γονέας του X και ο γονέας του Y είναι ξαδέλφια.

## Κανόνας 9

uncle(X,Y) :- male(X),sibling(X,Parent),parent(Parent,Y),X\=Y.

Ο X είναι θείος του Y αν είναι άντρας και ο Γονέας Parent του Y είναι αδελφός του .

```
?- uncle(X,esteban).
X = jonah ,
```

## Κανόνας 10

aunt(X,Y) :- female(X),sibling(X,Parent),parent(Parent,Y),X\=Y.

Ο X είναι θεία του Y αν είναι γυναικά και ο Γονέας Parent του Y είναι αδελφός του .

```
?- aunt(X,esteban).
X = angelique ,
```

# Θέμα 2

## Κατηγόρημα 1

/*sumlist(List,S).  %  List = λίστα αριθμών, S = άθροισμά τους*/
Επιστρέφει το άθροισμα S όλων τον στοιχείων μιας λίστας List.

```
/*-----------------------------------------*/
sumlist([ ],0).
sumlist([H|T],S) :- sumlist(T,S1),S is S1 + H .
/*-----------------------------------------*/
```

Το άθροισμα των στοιχείων την κενής λίστας είναι 0 (τερματική συνθήκη).
Το άθροισμα μια μη κενής λίστας ισούται με το άθροισμα του πρώτου στοιχείου Η συν το άθροισμα της λίστας που αποτελείται από όλα τα υπόλοιπα στοιχεία Τ.
Και αναδρομικά βρίσκουμε τη λύση.
παραδείγματος χάρη :

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

?- sumlist([11,45,83,95,12],X).
X = 246.
```

## Κατηγόρημα 2

/*multlist(List,M).  %  List  = λίστα αριθμών, M  = γινόμενό τους*/
Επιστρέφει το γινόμενο M όλων τον στοιχείων μιας λίστας List.

```
/*-----------------------------------------*/
multlist([],1).
multlist([H|T],M) :- multlist(T,M1),M is M1 * H .
/*-----------------------------------------*/
```

Το γινόμενο των στοιχείων την κενής λίστας είναι 1 (τερματική συνθήκη).

Το γινόμενο μια μη κενής λίστας ισούται με το γινόμενο του πρώτου στοιχείου Η επιώ το γινόμενο της λίστας που αποτελείται από όλα τα υπόλοιπα στοιχεία Τ.
Και αναδρομικά βρίσκουμε τη λύση.
παραδείγματος χάρη :

```
?- multlist([1,2,3,4,5],X).
X = 120.

?- multlist([11,45,83,95,12],X).
X = 46836900.
```

Κατηγόρημα 3

```
/*listlength(List,L).  %   List = λίστα, L = μήκος λίστας*/
Βρίσκει το μέγεθος μιας λίστας List


/*-----------------------------------------*/
listlength([],0).
listlength([H|T],L) :- listlength(T,L1), L is L1 + 1 .
/*-----------------------------------------*/
```

Το μέγεθος της κενής λίστας είναι 0 (τερματική συνθήκη).
Το μέγεθος μια μη κενής λίστας ισούται με 1 συν το μέγεθος της υπόλοιπης λίστας Τ .
Και αναδρομικά βρίσκουμε τη λύση.
παραδείγματος χάρη :

```
?- listlength([11,45,83,95,12],X).
X = 5.

?- listlength([11,45,83,95,12,65,12,56],X).
X = 8.
```

Κατηγόρημα 4

```
/*first_element(List,F).   %   List     = λίστα, F   =
πρώτο στοιχείο της λίστας*/
Επιστρέφει το πρώτο στοιχείο της λίστας

/*-----------------------------------------*/
first_element([],"List is Empty").
first_element([H|T],F) :-  F is H .
/*-----------------------------------------*/
```

Αν έχουμε την κενή λίστα επιστρέφει το string: “List is Empty”.
Στην λίστα List που αποτελείται από το Head H και Tail T το πρώτο στοιχείο είναι το H.
παραδείγματος χάρη :

```
?- first_element([4,5],X).
X = 4.

?- first_element([12,4,5],X).
X = 12.
```

Κατηγόρημα 5

```
/*last_element(List,L).    %   List     = λίστα, L   =
Τελευταίο στοιχείο της λίστας*/
Επιστρέφει το τελευταίο στοιχείο της λίστας
/*-----------------------------------------*/
last_element([L],L).
last_element([_|X],L):- last_element(X,L).
/*-----------------------------------------*/
```

Αν έχουμε την λίστα με ένα στοιχείο L επιστρέφει το L .
Στην λίστα που αποτελείται από οτιδήποτε _ και Tail X το τελευταίο στοιχείο της είναι το τελευταίο στοιχείο της λίστας X.
παραδείγματος χάρη :

```
?- last_element([1,2,3],X).
X = 3 .

?- last_element([1,2,6],X).
X = 6 .
```

Κατηγόρημα 6

```
/*nth_element(List,N,E).   %   Επιστρέφει στο E το N-οστό     στοιχείο της λίστας List*/


/*-----------------------------------------*/
nth_element([X|_],X,1).
nth_element([_|L],X,K) :- K > 1, K1 is K - 1, nth_element(L,X,K1).
/*-----------------------------------------*/
```

Δουλεύουμε αναδρομικά:
Το στοιχείο στην θέση 1 είναι το Head X που ακολουθείται από οτιδήποτε.
Το στοιχείο στην θέση K της λίστας με Tail L είναι το K-1 στοιχείο της λίστας L.
παραδείγματος χάρη :

```
?- nth_element([11,45,83,95,12,65,12,56],X,3).
X = 83 .

?- nth_element([11,45,83,95,12,65,12,56],X,7).
X = 12 .
```

Κατηγόρημα 7

```
/*element_position(List,E,N).  %   Επιστρέφει στο N την θέση του στοιχείου E της λίστας List*/


/*-----------------------------------------*/
element_position([X|Ys],X,1).
element_position([Y|Ys],X,N) :- element_position(Ys,X,K), N is K+1.
/*-----------------------------------------*/
```

Αν το στοίχοι X που ψάχνουμε είναι το Head της λίστας με Tail Ys τότε επιστρέφει 1.
Αν το στοίχοι X που ψάχνουμε είναι το Head Y και Tail Ys βρίσκεται στην θέση N αν βρίσκεται στην θέση K+1 της λίστας Ys.
παραδείγματος χάρη :

```
?- element_position([11,45,83,95,12,65,12,56],11,X).
X = 1 .

?- element_position([11,45,83,95,12,65,12,56],12,X).
X = 5 .
```

Κατηγόρημα 8

```
/*my_member(X,List).        %    X    είναι στοιχείο της
λίστας List*/

/*-----------------------------------------*/
my_member(X, [Y|Ys]) :- X = Y ; my_member(X, Ys).
/*-----------------------------------------*/
```

Εάν στοιχείο X ανήκει της λίστας είναι το Head Y της λίστας η ανήκει στην υπόλοιπη λίστα Ys.

παραδείγματος χάρη :

```
?- my_member(56,[11,83,56,34,9,77,12,91]).
true .

?- my_member(88,[11,83,56,34,9,77,12,91]).
false.
```

Κατηγόρημα 9

```
/*my_select(X,List,Rest)    %    X είναι στοιχείο της
λίστας List,      Rest      η υπόλοιπη λίστα*/

/*-----------------------------------------*/
del(X,[X|T],T).
del(X,[H|T],[H|R]) :- del(X,T,R).
/*-----------------------------------------*/
```

Αν το στοιχείο X είναι Head της λίστας τότε επιστρέφει το Tail T. Προσπερνά την λίστα μέχρι το X να είναι Head και το αφαίρει.

παραδείγματος χάρη :

```
?- del(7,[1,7,6,11,13,77],X).
X = [1, 6, 11, 13, 77] .

?- del(13,[1,7,6,11,13,77],X).
X = [1, 7, 6, 11, 77] .
```

Κατηγόρημα 10

```
/*my_select(X,List,Rest)    %    X είναι στοιχείο της
λίστας List,      Rest      η υπόλοιπη λίστα*/
/*-----------------------------------------*/
merge_list([],L,L ).
merge_list([H|T],L,[H|M]):- merge_list(T,L,M).
/*-----------------------------------------*/
```

Η ένωση της κενής λίστας με την λίστα L είναι η λίστα L.

Η ένωση της λίστας L με την λίστα [H.M] είναι μια λίστα με Head H και Tail το T είναι η ένωση L και M.
παραδείγματος χάρη :

```
?- merge_list([a,b,c,d], [e,f,g], R).
R = [a, b, c, d, e, f, g].

?- merge_list([], [], R).
R = [].

?- merge_list([1], [1], R).
R = [1, 1].
```

# Θέμα 3

Αναπαριστάμε τον χάρτη σαν μια λίστα από λίστες

map1([[0, 0, 0, 0, 0], [1, 0, 0, 1, 1], [1, 1, 0, 1, 1], [1, 1, 1, 1, 1], [1, 1, 1, 1, 1]])

είναι ο χάρτης :

```
% X     1 2 3 4 5     Y
map1([
       [0,0,0,0,0],   %1
       [1,0,0,1,1],   %2
       [1,1,0,1,1],   %3
       [1,1,1,1,1],   %4
       [1,1,1,1,1]    %5
]).
```

## 1ο :

```
/*-----------------------------------------*/
```

nth_element([X|H],X,1).

nth_element([H|L],X,K) :- K > 1, K1 is K - 1, nth_element(L,X,K1).

```
/*-----------------------------------------*/
```

Επιστρέφει το ν-ιοστο στοιχείο μιας λίστας , εάν ψάχνουμε το πρώτο στοιχείο της λίστας τότε είναι το Head της λίστας ,διαφορετικά το βρίσκουμε αναδρομικά

```
?- nth_element([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]],1,X).
X = [0, 0, 0, 0, 0].

?- nth_element([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]],3,X).
X = [1, 1, 0, 1, 1].
```

2ο :

```
/*-----------------------------------------*/

nth_map(Map, C, L, Element) :-
    nth_element(Map, L, Line),
    nth_element(Line, C, Element), !.
/*-----------------------------------------*/
```

Βρίσκει και επιστρέφει στο Element το στοιχείο του χάρτη στην στήλη C και γραμμή L αρχειακά χρησιμοποιώντας τον κανόνα nth_element αρχικά θα βρούμε σε ποια λίστα βρισκόμαστε στον χάρτη .

Και μετα θα βρούμε ζητούμενο στοιχείο στην στήλη L του στοιχείου της λίστας Y.

```
?- nth_map([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]], C, L, E).
C = L, L = 1,
E = 0.

?- nth_map([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]], 2, 3, E).
E = 1.

?- nth_map([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]], 1, 2, E).
E = 1.

?- nth_map([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]], 3, 4, E).
E = 1.

?-
```

3ο :

```
/*-----------------------------------------*/

map_position(Map, Element, C, L) :-

    element_position(Map, Line, L),

    element_position(Line, Element, C).

/*-----------------------------------------*/
```

Χρησιμοποιεί την element_position Βρίσκει την θέση X Y του στοιχείου Element στον Χάρτη Map .

```
?- map_position([[0,0,0,0,0], [1,0,a,1,1],[1,1,0,1,1],[1,m,1,1,1],[1,1,1,1,1]],a,X,Y).
X = 3,
Y = 2 .

?- map_position([[0,0,0,0,0], [1,0,a,1,1],[1,1,0,1,1],[1,m,1,1,1],[1,1,1,1,1]],m,X,Y).
X = 2,
Y = 4 .

?- map_position([[0,0,0,0,0], [1,0,a,1,1],[1,1,0,1,1],[1,m,1,1,1],[1,1,1,1,1]],1,X,Y).
X = 1,
Y = 2 .
```

4ο :

```
/*-----------------------------------------*/

printlist([]) :- nl. !.
printlist([X|Xs]) :- write(X), write(' '), printlist(Xs).
/*-----------------------------------------*/
```

Τυπώνει όλα τα στοιχεία μιας λίστας το ένα μετα το άλλο

Εάν η λίστα είναι κενή αλλάζουμε σειρά και τελειώνει την διαδικασία

Εάν η λίστα είναι Head X και Tail Xs γράφουμε το X και « » και καλούμαι την printlist στην Xs

```
?- printlist([[0,0,0,0,0], [1,0,a,1,1],[1,1,0,1,1],[1,z,1,1,1],[1,1,1,1,1]]).
[0,0,0,0,0] [1,0,a,1,1] [1,1,0,1,1] [1,z,1,1,1] [1,1,1,1,1]
true.


?- printlist([a,b,c,d,e,f]).
a b c d e f
true.
```

5ο :

```
/*-----------------------------------------*/

printmap([]) :- nl , !.

printmap([Line|Rest]) :- printlist (Line), printmap(Rest).

/*-----------------------------------------*/
```

Χρησιμοποιεί την  για να τύπωση μια μια τοις γραμμές του χαρτί

```
?- printmap([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]]).
0 0 0 0 0
1 0 0 1 1
1 1 0 1 1
1 1 1 1 1
1 1 1 1 1

true.

?-
```

6ο :

```
/*-----------------------------------------*/

map_maxXY(M, X, Y):-
    first_element(M, F),
    listlength(F, X),
    listlength(M, Y).

/*-----------------------------------------*/
```

Χρησιμοποιεί της first_element Και listlength για να βρει τα οροια του χαρτί

```
?- map_maxXY([[0,0,0,0,0], [1,0,0,1,1],[1,1,0,1,1],[1,1,1,1,1],[1,1,1,1,1]], 5, 5).
true.
```

# Θέμα 4

Έχουμε τον χάρτη :

| | | | | | | |
|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |

Ο οποίος θα εκφραστεί ως μια λίστα από λίστες :

```
/*Column,X 1 2 3 4 5 6 7   Line,Y*/
map(     [[0,1,0,1,1,1,0],     % 1
           [0,1,0,0,0,0,0],    % 2
           [0,1,0,1,0,0,1],    % 3
           [0,0,0,0,0,1,1],    % 4
           [0,1,1,1,1,1,1],    % 5
           [0,0,1,0,0,1,1]% 6
           ]).
```

Οι έγκυρες κινήσεις είναι :

- Πανό move((C1,L1), (C2,L2)) :- C2 is C1,  L2 is L1-1.
- Πάνω Δεξιά move((C1,L1), (C2,L2)) :- C2 is C1+1, L2 is L1-1.
- Δεξιά move((C1,L1), (C2,L2)) :- C2 is C1+1, L2 is L1.
- Κάτω Δεξιά move((C1,L1), (C2,L2)) :- C2 is C1+1, L2 is L1+1.
- Κάτω move((C1,L1), (C2,L2)) :- C2 is C1,  L2 is L1+1.
- Κάτω Αριστερά move((C1,L1), (C2,L2)) :- C2 is C1-1, L2 is L1+1.

- Αριστερά move((C1,L1), (C2,L2)) :- C2 is C1-1, L2 is L1.
- Πανό Αριστερά move((C1,L1), (C2,L2)) :- C2 is C1-1, L2 is L1-1.

Εφόσον το X και Y μένουν στα επιτρεπτά όρια Δηλ :

0 < X <= 7 και 0 < Y <=6

**1ος Κανόνας :** Επιστρέφει το  στοιχείο της στήλης C και γραμμής L λίστας του map.

```
/*-----------------------------------------------*/
nth_map(Map, C, L, Element) :-
        nth_element(Map, L, Line),
        nth_element(Line, C, Element), !.
/*-----------------------------------------------*/
```

```
?- nth_map([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 1, 2, Element)
|    .
Element = 0.

?- nth_map([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 2, 1, Element)
|    .
Element = 9.

?- nth_map([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 2, 1, Element).
Element = 9.

?- nth_map([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 1, 2, Element).
Element = 0.

?- nth_map([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 3, 4, Element).
Element = 0.
```

**2ος Κανόνας :** Επιστρέφει την θέση του στοιχείου Element στον χάρτη δηλ. την Γραμμή και Στήλη που βρίσκεται.

```
/*-----------------------------------------------*/
        map_position(Map, Element, C, L) :-
                element_position(Map, Line, L),
                element_position(Line, Element, C).
/*-----------------------------------------------*/
```

```
?- map_position([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 9, C, L).
C = 2,
L = 1 .

?- map_position([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 3, C, L).
C = 4,
L = 1 .

?- map_position([[0,9,0,3,1,1,0],[0,6,0,0,0,0,0],[0,1,0,1,0,0,1],[0,0,0,0,0,1,1],[0,1,1,1,1,1,1],[0,0,1,0,0,1,1]], 1, C, L).
C = 5,
L = 1 ;
C = 6,
L = 1 ;
C = 2,
L = 3 ;
C = 4,
L = 3 ;
C = 7,
L = 3 ;
C = 6,
L = 4 ;
C = 7,
L = 4 ;
C = 2,
L = 5 ;
C = 3,
L = 5 ;
C = 4,
L = 5 ;
C = L, L = 5 ;
C = 6,
L = 5 ;
C = 7,
L = 5 ;
C = 3,
L = 6 ;
C = L, L = 6 ;
C = 7,
L = 6 ;
false.
```

**3ος κανόνας :** Εξετάζει αν το στοιχείο στην στήλη C και γραμμή L είναι μέσα στα όρια.

```
/*-----------------------------------------------*/
in_limits((C,L)) :-
        map(Map), length(Map,MaxL),
        Map=[Line|_],length(Line, MaxC),
        1=<C, C=<MaxC, 1=<L, L=<MaxL.
/*-----------------------------------------------*/
```

```
?- in_limits((2,2)).
true.

?- in_limits((7,1)).
true.

?- in_limits((9,1)).
false.
```

**4ος κανόνας :** Εξετάζει αν μια θέση (X,Y) είναι μια έγκυρη θέση στον χάρτη map .

1. εξετάζει αν είναι μέσα στα όρια του χαρτί με την in_map και μετα
2. αν η θέση αυτή είναι 0 αφού έχουμε συμφωνήσει ότι με αλλά σύμβολα συμβολίζουμε τον τοίχο με χρήση της nth_map.

```
/*-----------------------------------------------*/
valid_square(Y) :- in_limits(Y), empty_square(Y).
/*-----------------------------------------------*/
```

```
?- valid_square((2,2)).
false.

?- valid_square((7,1)).
true.

?- valid_square((9,1)).
false.
```

**5ος κανόνας :** Εξετάζει αν υπάρχει μια έγκυρη κίνηση από μια θέση X σε μια θέση Y στον χάρτη map και επιστρέφει true ή false.

```
/*-----------------------------------------------*/
validmove(X,Y) :- move(X,Y), valid_square(Y).
/*-----------------------------------------------*/
```

```
?- validmove((1,1),(1,2)).
true .

?- validmove((1,1),(2,1)).
false.

?- validmove((3,4),(4,4)).
true .
```

**6ος Κανόνας :**

Επιστρέφει το μονοπάτι αν υπάρχει από το Start στο Finish.

```
/*-----------------------------------------------*/
findpath :-
        start(S),finish(F),
        path(S,F,[S],P), print_list(P).

path(X,Y,_,[X,Y]) :- validmove(X,Y).
path(X,Y,Visited,[X|Rest]) :-
        validmove(X,Z),
```

```
        not(member(Z,Visited)),
        path(Z,Y,[Z|Visited],Rest).
/*-----------------------------------------------*/
```

Οπού συμβολίζουμε με :

- Start ένα έγκυρο σημείο εκκίνησης.
- Finish ένα έγκυρο σημείο τερματισμού.
- Visited λίστα με τα σημεία που έχουμε επισκεφτεί.

Για Start = (1,1) και Finish = (6,4) έχουμε :

```
?- findpath.
false.
```

Για Start = (1,1) και Finish = (3,1) έχουμε :

```
?- findpath.
1,1
1,2
1,3
2,4
3,3
3,2
3,1
true .
```

Για Start = (1,1) και Finish = (7,1) έχουμε :

```
?- findpath.
1,1
1,2
1,3
2,4
3,3
3,2
3,1
4,2
5,2
6,2
7,1
true .
```