1 The Story So Far ...

- Explanation in logic.
- Definitions and facts.
- Predicates / Rules.
- Translation from a logic construction.
- Conjunction and Disjunstion.
- Nested rules.
- Rule Syntax (Anatomy of a predicate).
- Atoms, Variables.
- Anonymous Variables and PlaceHolders.
- Yes/No Queries.
- Search-a-database Query.

Any problems with these ?

2 Programming Guidelines

- Meaningful names for variables and predicates.
- simple predicates.
- keep database seperate.
- Put plenty of comments.
- Do not use variable names like _(number)

3 More Examples

Two people are neighbours if they live on the same street but in different houses.

```
neighbour( Person_A, Person_B) :-
  address( Person_A, Street, Num),
  address( Person_B, Street, Num1).
```

Translate the following into Prolog rules :

1. Nothing is good and bad at the same time.
2. X lives if X is a human, if he/she breathes and if his/her pulse-rate is greater than zero.
3. I hate McDonalds.
4. Liam plays all the songs that the Beatles played without a keyboard.
5. At the restaurant I can have cheese with apple or bread or coffee and chocolate with donuts.

4 A program ... FINALLY!!

```
/* hifi(Name, Rms, Tapes, Cds, Turntable, Price) */
hifi(sony, 28, 2,1,1,279).
hifi(aiwa, 25, 2,1,1,249).
hifi(sanyo,12, 2,3,1,219).
hifi(pioneer,40,2,1,0,379).
hifi(akai, 12, 2,3,1,379).

preferable( Hifi, Price) :-
 hifi(Name, Rms, Tapes, Cds, Turntable, Price),
 Rms > 25.
preferable( Hifi, Price) :-
 hifi(Name, Rms, Tapes, Cds, Turntable, Price),
 Tapes > 1.

will_do( Hifi, Price) :-
 hifi(Name, Rms, Tapes, Cds, Turntable, Price),
 Tapes > 0.
will_do( Hifi, Price) :-
 hifi(Name, Rms, Tapes, Cds, Turntable, Price),
 Rms > 10.
```

Program is constructed from :

Procedures : set of clauses about the same relation, containing the same predicate in their heads.

Price is a shared variable - restricts range.

Procedures are :
hifi/6
preferable/2
will_do/2

The Program constitutes the **formal definition** *of the problem, it is not yet the solution. The solution is what occurs when we ask Prolog a question.*

preferable(Hifi, Price), Price ¡ 200.
no

preferable(Hifi, Price), Price ¡ 300.
Hifi=sony, Price=279
Hifi=akai, Price=279)

5 Compound Terms and Structures

- *Most real world objects constitute a combination of items or atoms put together.*
- *To denote a structure, we chose a functor and its arguments.*

An example, a menu :

lunch(Soft_drink, Main_meal, Donut, Beverage)

eg : lunch(diet_coke, pizza, donut, coffee)

donut(jam_donut, ring_donut, choc_donut, caramel_donut, custard_donut)

With lunch now becoming a mouthful :

lunch(Soft_drink, Main_meal,
donut(jam_donut, ring_donut, choc_donut, caramel_donut, custard_donut),
Beverage).

etc ...

As structures get more complex, it is usually better to start representing them as **Trees***, so that the relationships between sub-structures become more clear.*

More Lists

A non-empty list consists of a HEAD and a TAIL.

The FIRST ELEMENT is said to be the HEAD of the list and the rest constitutes the TAIL.

We can make this explicit, by using the $||notation, forinstance:$

```
[1,2,3] = [1 | [2,3]]

["One", "Two", "Three"] "One"  ["Two","Three"]

[145] 145 []

[] undefined undefined

[[1,2],[3,4,5],[]] [1,2] [[3,4,5],[]]

[a] a []
```

List is a very important and useful data structure within Prolog, and extensively used.

6 Equating Lists

```
List1 : [X,Y,Z]
List2 : [4,5,6]

List1 = List2 X=4, Y=5, Z=6

List1 : ["hello"]
List2 : [X|Y]

List1 = List2  X = "hello", Y=[]

List1 : [1,2,3,4]
List2  : [X,Y|Z]

List1 = List2 X=1, Y=2, Z=[3,4]

List1 : [1,X]
List2 : [Y,3]

List1 = List2 Y=1, X=3
```

7 List Membership

```
member(X,L)
member(b,[a,b,c]).

One :
member(X,[X | Tail]).

member(X, [Head | Tail]) :-
 member(X, Tail).

Two :
member(X,[X|_]).

member(X, [_|Tail]) :-
 member(X,Tail).
```

member(2,[1,4,2,3,5]).

Ist Clause : No match
2nd Clause : Member if member of [4,2,3,5]
Ist Clause : Satisfied
Hence Answer = yes

8 Alternate List Representation

- *Possible to represent lists like other Prolog structures using a special functor '.' (dot).*
- **.(Head, Tail)**
- *For instance, a list like :*

miles, evans, holliday, parker, getz

can be expressed as :

.(miles, .(evans, .(holliday, .(parker, .(getz,[])))))

```
 The  dot notation for internal representation

List = [ a, b, c, d \| Tail ]

then extend Tail to
Tail = [ e | Tail1 ]

List = [a, b,c,d,e \| Tail1]

Hence, arbitary extension of lists.
```