COMP307

- Class Rep
- Assignment 1 is handed out.
- Tutorial this week: Friday 12:00-12:50 HM LT 002
- Help desks this week:

  Wednesday 3-4, CO238, Friday 2-3 CO239

Prolog Materials

SWI-Prolog Manual: 2.1, 2.5, …

edit a file using emacs or other editor: myfile.pl

command line "prolog" or "swi-prolog"

(or use it in emacs see 2.5)

load a file and compile ?-[myfile].

Learning Prolog online course

Prolog books:

- The Art of Prolog (Sterling),
- Prolog by Example (Helder Coelho, Jose C. Cotta)
- Prolog programming for Artificial Intelligence (Ivan Bratko)

Prolog (II)

Recursion in Prolog

Data Structures

- Structure
- List

"while" in prolog

"Tail Recursion" instead of "while"

Base case: stop

Normal case:

do one step

change the arguments

call the same clause using different arguments

```
print_stars(0):-!.
print_stars(N):-
    write('*'),
    N1 is N-1,
    print_stars(N1).

|?-print_stars(5).
```

Recursion in Java

- n! = 1 * 2 * 3 *...* n

n! = n * (n-1)!

0!=1

- Recursive method in Java: a method that calls itself

```
public int factorial(int n) {
    if (n == 0)
          return 1;
    else {
          int f = factorial (n-1);
          return n * f;
     }
}
```

Recursion in Prolog

```
factorial(0, 1):-!.
factorial(N, X):-
  N1 is N-1,
  factorial(N1, X1),
  X is N * X1.
```

```
|?-factorial(4, X).
X=24.
```

Base case: do something

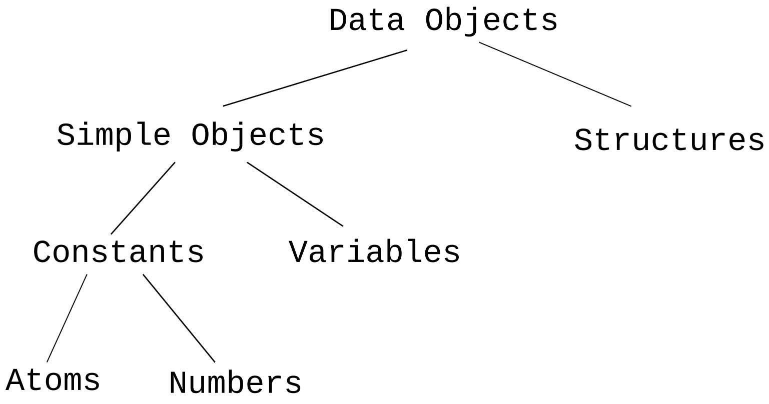
Normal case:

change the arguments

call the same clause using different arguments

do one step

Data Objects



Structured terms (Structured Objects

```
person(john, smith, date_birth(1, may, 1968)).

course(comp307, lectures(hm104, sharon)).

picture(rectangle(10, 20)).
picture(rectangle(width(10), height(20))).
picture(rectangle(top_left(100, 100), 10, 20)).
```

Structure: functor(components)

components can be other structures

Structured terms can be used to represent objects, trees, and other data structures.

List examples

```
names(comp307, [john, mary, helen]).
seat_plan(comp307, [[john, mary, helen], [jim,
  tim]]).
marks([80.5, 90, 95]).
marks([john, 80], [mary, 90], [helen, 95]).
list_example([1, 2, 3, a, b, [c, d]]).
list_example([ ])

picture([rectangle(top_left(100, 100), 10, 20),
  circle(center(10, 10), 50)]).
```

List

- List is a sequence of any numbers of terms.
- List is a special structure

  .(Head, Tail)

  [a, b, c, d]   .(a, [b, c, d])   .(a, .(b, .(c, .(d, [ ]))))
- The length of the list is not fixed
- Sequential access only
- Can only add/delete elements to/from the beginning of a
- The elements of the list
  - can be anything (even other lists),
  - can be different type

[Head|Tail]

[Head|Tail]   .(Head, Tail)

Head is the first element of the list,

Tail is the rest of the list (**Tail is a list itself**).

`[Head|Tail]`	`[a, b, c]`	Head=a, Tail=[b, c]
`[Head|Tail]`	`[b,c]`	Head=b, Tail=[c]
`[Head|Tail]`	`[c]`	Head=c, Tail=[]
[Head|Tail]	[ ]	do not match
[Head]	[e]	Head =e
[Head]	[ ]	do not match

Tail recursion to handle lists

|?-print_list([a, b, c]).

```
print_list([]).
print_list([H|T]):-
     write(H),
     print_list(T).
```

Tail Recursion to handle lists

?- member(1, [1, 2, 3]).

?-member(3, [1, 2, 3]).

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

Recursion and lists

|?-list_length([a, b, c], X).

```
list_length([], 0).
list_length([Head|Tail], X):-
     list_length(Tail, X1),
     X is X1 +1.
```

Using List to collect results: recursion (rope-ladder)

```
|?-list_scaleup([40, 60, 80], X).

list_scaleup([], []).
list_scaleup([H|T], L):-
  list_scaleup(T, TT),
  HH is H+10,
  L=[HH|TT].

list_scaleup([], []).
list_scaleup([H|T], [HH|TT]):-
  HH is H+10,
  list_scaleup(T, TT).
```

Using List to collect results: recursion (rope-ladder)

|?-intersection([a, b, c], [b, c, d], X).

% intersection(+Xs, +Ys, -Result)

```
intersection([], _, []).
intersection([X|Xs], Ys, [X|Zs]) :-
  member(X, Ys), !,
  intersection(Xs, Ys, Zs).
intersection([X|Xs], Ys, Zs) :-
  \+ member(X, Ys),
  intersection(Xs, Ys, Zs).
```