

Les entrées–sorties

- ▶ Ecriture sur l'écran ou dans un fichier
- ▶ Lecture à partir du clavier ou d'un fichier
- ▶ Affichage de termes :
 - * `write(1+2)` affiche `1+2`
 - * `write(X).` affiche la valeur courante de `X` sur le flot de sortie courant (par défaut l'écran),
 - * `nl` permet de passer à la ligne suivante.
 - * `writeln(X) :- write(X), nl.`
 - * `tab` tel que `tab(N)` affiche N espaces

Affichage

▶ Affichage de termes (suite) :

- * *display/1* agit comme *write/1* mais en affichant la représentation sous forme d'arbre

- * Ex :

write(3+4), nl, display(3+4), nl.

Affiche :

$3+4$

$+(3,4)$

YES

Lecture

▶ Lecture de termes :

- * *read/1* admet n'importe quel terme en argument.
 - * Il lit un terme au clavier et l'unifie avec son argument. Le terme lu doit être obligatoirement suivi d'un point. Certains systèmes Prolog affichent un signe d'invite lorsque le prédictat *read/1* est utilisé.

- * Exemple :

```
?- read(X).
```

```
: a(1,2).
```

```
YES {X = a(1,2)}
```

Exemple

```
age(X, Y) :-  
    write('Give the age of '),  
    write(X), write(': '),
    read(Y).  
?- age(teddy, Z).  
Give the age of teddy: 22.  
Z = 22  
Yes
```

```
?- age(teddy, 22).  
Give the age of teddy: 23.  
No  
?- read(abc).  
:23.  
No  
?- read(X + Y).  
:2 + 3.  
X = 2  
Y = 3  
Yes
```

Un autre exemple

lire des expressions arithmétiques, les évaluer et les imprimer jusqu'à ce que l'utilisateur rentre « fin » au clavier.

```
calculateur :- repeat,      % boucle
              read(X),    % lecture expression
              eval(X,Y),   % évaluation
              write(Y), nl, % affichage
              Y = fin, !.  % condition d'arrêt
eval(fin, fin) :- !.          % cas particulier
eval(X, Y) :- Y is X.        % calcul d'expressions
```

Le repeat

Le prédictat *repeat* laisse toujours un point de choix derrière lui.

repeat.

repeat :- repeat.

Exemple d'utilisation :

?- *calculateur.*

: 2+3 .

5

: 3+2*4 -1.

10

: *fin.*

fin

YES

Autre exemple avec repeat

```
test :- repeat,  
        write('SVP, entrer un nombre'),  
        read(X),  
        (X=:=42).
```

Ouvrir un fichier

- En écriture :
 - * mode *write* : son contenu est effacé avant que Prolog y écrive.
 - * mode *append* : Prolog écrira à partir de la fin du fichier.
- Ouverture d'un fichier : prédicat *open/3*
 - * argument 1 : nom du fichier
 - * argument 2 : mode d'ouverture *write*, *append* ou *read*
 - * argument 3 : variable qui va recevoir un identificateur de fichier appelé *flux* ou *stream*.

Lire et écrire

- ▶ Tous les prédicats *read*, *write* et autres vus auparavant admettent un second argument : le flux identifiant le fichier.
 - * Ex : *write(Flux, X)*. où Flux est un identificateur de fichier
 - * Ex : *read(Flux,X)*, *get(Flux, X)*, *get0(Flux,X)*
 - * Fermeture du fichier : prédicat *close/1* qui prend en argument le flux associé au fichier.

Exemple

```
écrire(T) :-  
    open(' test.pl ', append, Flux), (*ouverture*)  
    write(Flux, T), nl(Flux),      (*écriture*)  
    close(Flux).                  (*fermeture*)
```

Les flots d'entrée et de sortie

- ▶ *see(Filename)*, le fichier est l'entrée courante.
- ▶ *seen*. La console redevient l'entrée courante.
- ▶ *tell(Filename)*, le fichier est la sortie courante.
- ▶ *told*. La console redevient la sortie courante.

Les caractères

- ▶ *put(CodeASCII)* : imprime le caractère correspondant au code ASCII.
- ▶ *get0(Code)* : unifie la variable avec le code ASCII du caractère entré.
- ▶ *get(Code)* : même chose que get0, mais saute par-dessus les espaces.

Exemple interactif

```
capitale(ontario,toronto).
capitale(quebec,quebec).
capitale(cb,victoria).
capitale(alberta,edmonton).
capitale(terre-neuve,st-jean).
capitale(nouvelle-ecosse,halifax).
capitale(saskatchewan,regina).
capitale(manitoba,winnipeg).
capitale(nouveau-brunswick,fredericton).
capitale(ipe,charlottetown).
start:-write('Les Capitales du Canada'),nl,demander.
demander:-write('Province? '),read(Province),reponse(Province).
reponse(stop):-write('merci'),nl.
reponse(Province):-capitale(Province,Ville),write('la capitale de '),
                  write(Province),write(' est '),write(Ville),nl,nl,demander.
```

Exemple (suite)

?- start.

Les Capitales du Canada
Province? ontario.

la capitale de ontario est toronto

Province? cb.

la capitale de cb est victoria

Province? stop.

merci

true .

Les Listes

- ▶ Comme en programmation fonctionnelle, la liste est une structure de donnée de base :
 - [1, 2, 3, 4]
 - [] la liste vide ;
 - [Head | Tail] la tête et le reste de la liste ;
 - [1, 2, "trois"] une liste de 3 éléments ;
 - [1, 2 | Tail] une liste d'au moins deux éléments.

Format Tête et Queue

?- [T | Q] = [1, 2, 3, 4].

T= 1,

Q= [2,3,4]

?- [1 | [2,3,4]] = L.

L= [1,2,3,4]

?- [1,2,3 | [4]] = L.

L= [1,2,3,4]

?- [T | Q] = [1].

T= 1,

Q= []

?- [T | Q] = [].

no

Exemple

Lire des caractères en créant une liste,
jusqu'à la fin d'une ligne (code 10)

```
readline(Line) :-  
    get0(Ch), readline(Ch, Line).
```

```
readline(10, []).  
readline(Ch, [Ch | RestOfLine]) :-  
    Ch \= 10,  
    get0(NextCh),  
    readline(NextCh, RestOfLine).
```

Construction de listes

`cons(X, Y, [X|Y]).`

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

?- `cons(X, Y, [1,2,3,4]).`
`X= 1,`
`Y= [2,3,4]`

?- `cons(1, [2,3,4], [1,2,3,4]).`
`yes`

Concaténation de listes

```
notre-append([],Y,Y).
```

```
notre-append([A|B],Y,[A|W]) :- notre-append(B,Y,W).
```

```
?- notre-append([1,2], [3,4], L).
```

```
L= [1,2,3,4]
```

```
?- notre-append(X, [3,4], [1,2,3,4]).
```

```
X= [1,2]
```

```
?- notre-append([1,2], [3,4], [1,2,3,4]).
```

```
yes
```

Inversion de listes, version 1

```
notre-reverse([],[]).
```

```
notre-reverse([H|T],L) :- notre-reverse(T,LL),  
                      notre-append(LL,[H],L).
```

```
?- notre-reverse([1,2,3,4],L).
```

```
L= [4,3,2,1]
```

```
?- notre-reverse(L,[1,2,3,4]).
```

```
L= [4,3,2,1]
```

Inversion de listes, version 2

```
renverser([],L,L)-!.
```

```
renverser([H|T],L,R):- renverser(T,[H|L],R).
```

```
notre-reverse(L,R) :- renverser(L,[],R).
```

Sans la coupe, il y aurait une boucle infinie après la première solution de:

```
?- notre-reverse(L,[1,2,3,4]).
```

Appartenance à une liste

notre-member(X,[X|L]).

notre-member(X,[Y|L]) :- notre-member(X,L).

Longueur d'une liste

notre-length([],0).

notre-length([X|L],N) :- notre-length(L,NN), N is NN+1.

Insertion dans une liste

```
notre-insert(A,L,[A|L]).
```

```
notre-insert(A,[X|L], [X|LL]) :- notre-insert(A,L,LL).
```

```
?- insert(c, [a, b], L).
```

```
L = [c, a, b] ;
```

```
L = [a, c, b] ;
```

```
L = [a, b, c] ;
```

```
no
```