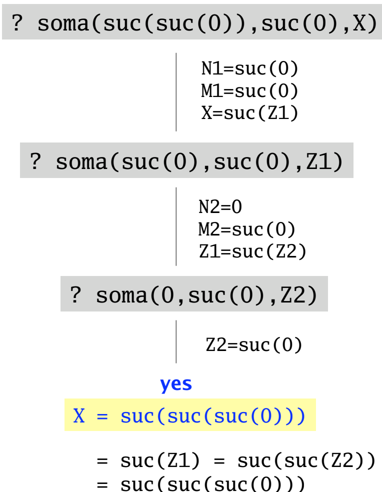
Resolução: Uma solução para a soma de números naturais

```
soma(0,N,N).
soma(suc(N),M,suc(Z)) :- soma(N,M,Z).
```

Exemplo de uma árvore de procura



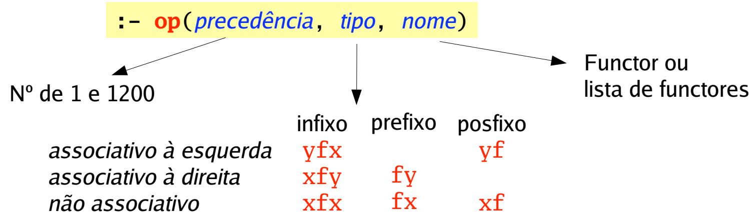

```
= suc(Z1) = suc(suc(Z2))
= suc(suc(suc(0)))
```


Operadores

Para conveniência de notaçao, o Prolog permite declarar functores unários ou binários como operadores *prefixos*, *posfixos* ou *infixos*, associando-lhes ainda uma precedência.

A declaração de operadores faz-se através da directiva

:- **op**(*precedência*, *tipo*, *nome*)



	infixo	prefixo	posfixo
associativo à esquerda	yfx		yf
associativo à direita	xfy	fy	
não associativo	xfx	fx	xf

Exemplos:

```
:- op(500,yfx,[+,-])
:- op(400,yfx,*)
```

```
:- op(300,fy,nao)
```

`a-d+b*c` , `(a-d)+(b*c)` e `+(-(a,d),*(b,c))` são termos equivalentes

`nao nao p` , `nao (nao p)` e `nao(nao(p))` são termos equivalentes

= operador de unificação

O operador infixo = establece uma relação de unificação entre dois termos.

T1 = T2 sucede se *o termo T1 unifica com o termo T2*

Dois termo T1 e T2 **unificam** se existir uma substituição θ que aplicada aos termos T1 e T2 os torne literalmente iguais. Isto é, T1θ == T2θ .

- Dois átomos (ou números) só unificam se forem exactamente iguais.
- Dois termos compostos unicam se os functores principais dos dois termos forem iguais e com a mesma aridade, e os argumentos respectivos unificam.
- Uma variável unifica com qualquer termo (gerando a substituição respectiva).

Exemplo:

```
| ?- data(23,maio,1998) = data(23,maio,1998).
yes
| ?- data(23,maio,1998) = data(23,maio,Ano).
Ano = 1998 ?
yes
| ?- data(23,maio,1998) = data(15,maio,Ano).
no
| ?- data(23,maio,1998) = data(X,maio,X).
no
```


"Occurs-Check"

Exemplo:

```
| ?- hora(10,30) = hora(X,Y).
X = 10,
Y = 30 ?
yes
| ?- hora(H,30) = hora(10,M).
H = 10,
M = 30 ?
yes
| ?- hora(H,30,12) = hora(X,Y,X).
H = 12,
X = 12,
Y = 30 ?
yes
| ?- hora(H,30,12) = hora(X,X,X).
no
| ?- X = f(X).
X = f(f(f(f(f(f(f(f(f(f(...)))))))))) ?
yes
```

O Sicstus Prolog permite unificar uma variável com um termo em que essa variável ocorre, permitindo assim a criação de *termos cíclicos*. Formalmente isto não é desejável mas o teste de ocorrência não é feito por razões de eficiência.

Operadores aritméticos

Alguns operadores aritméticos do Sicstus Prolog (ver *User's Manual*):

`+  -  /  *  //  mod  abs  sign  gcd  min  max  round  truncate`

`sin  cos  tan  cot  asin  acos ...  sqrt  log  exp  **  ...`

Estes **operadores** são **simbólicos**, permitem construir expressões aritméticas, mas não efectuam qualquer cálculo.

Exemplo:

```
| ?- 5 = 4+1.
no
| ?- 3+X = 5.
no
```

```
| ?- A = 3 mod 2.
A = 3 mod 2 ?
yes
| ?- X-7 = 3*4-7.
X = 3*4 ?
yes
```

O **cálculo aritmético** é efectuado utilizando os seguintes predicados aritméticos pré-definidos:

`=:=   =\=   <   >   =<   >=`   *Comparam os valores das expressões numéricas.*

`Z is expressão`   *A expressão é calculada e o seu resultado unifica com Z. Se a expressão não for numérica a cláusula falha.*

Exemplos:

```
| ?- round(3.5) =:= round(3.9).
yes
| ?- abs(-5*min(1,7)) < 7//2.
no
```

```
| ?- X is gcd(20,15).
X = 5 ?
yes
| ?- Y is 5**2.
Y = 25.0 ?
yes
```

```
% tamanho(?L,?N)   N é o comprimento da lista L
tamanho([],0).
tamanho([_|T], Z) :- tamanho(T,X), Z is X+1.
```

```
% fib(+N,?Z)   Z é o número de Fibonacci de N
fib(0,0).
fib(1,1).
fib(N,X) :- N > 1, N1 is N-1, fib(N1,A),
            N2 is N-2, fib(N2,B), X is A+B.
```

Nota: Na documentação de um predicado costuma-se usar a seguinte notação:

- **+** *o argumento deve estar instanciado*
- **-** *o argumento deve ser uma variável não instanciada*
- **?** *o argumento pode, ou não, estar instanciado*
- **:** *o argumento tem que ser um predicado*

Comparação de termos

O Sicstus Prolog tem predicados pré-definidos para comparação (sintáctica) de termos (ver *User's Manual*).

termo1 == termo2   *Testa se os termos são literalmente iguais.*

termo1 \== termo2   *Testa se os termos não são literalmente iguais.*

Exemplos:

```
| ?- X == Y.
no
| ?- X*4 \== X*4
no
```

```
| ?- X = Y, X == Y.
Y = X ?
yes
| ?- X == Y, X = Y.
no
```

O Sicstus Prolog establece uma relação de ordem no conjunto de termos (ver o manual).
A comparação de termos de acordo com essa ordem pode ser feita com os operadores:

`@<   @>   @=<   @>=`

Exemplos:

```
| ?- abcd @< xyz.
yes
```

```
| ?- abcd(1,2) @=< xyz.
no
```


Exemplos:

```
| ?- hora(3+5,7-3) = hora(10-2,2*2).
no
| ?- hora(3+5,7-3) == hora(10-2,2*2).
no
| ?- [user].
% consulting user...
| :- op(700,xfx,igual).
| igual(hora(H1,M1),hora(H2,M2)) :- H1 =:= H2, M1 =:= M2.
|
% consulted user in module user, 0 msec 504 bytes
yes
| ?- hora(3+5,7-3) igual hora(10-2,2*2).
yes
| ?-
```

`[user].`   Permite acrescentar novas regras na base de conhecimento.
É um editor muito primitivo (linha a linha).
`^D` para sair do editor e carregar as novas regras.

Note que não se está a alterar nenhum ficheiro!

Exemplos:

```
% apaga todas as ocorrências de um dado elemento de uma lista
apaga([H|T],H,L) :- apaga(T,H,L).
apaga([H|T],X,[H|L]) :- H \== X, apaga(T,X,L).
apaga([],_,[]).
```

```
% ordenação de uma lista com o algoritmo insertion sort
isort([],[]).
isort([H|T],L) :- isort(T,T1), ins(H,T1,L).

ins(X,[],[X]).
ins(X,[Y|Ys],[Y|Zs]) :- X > Y, ins(X,Ys,Zs).
ins(X,[Y|Ys],[X,Y|Ys]) :- X =< Y.
```

Exercícios:

1. Defina os seguintes predicados sobre listas:
   a) `minimo/2` que produz o menor elemento presente numa lista.
   b) `somatorio/2` que cálcula o somatório de uma lista.
   c) `nesimo/3` que dá o elemento da lista na n-ésima posição
2. Defina um procedimento que ordene de uma lista segundo o algoritmo *quicksort*.

Uso de Acumuladores

Exemplo: Inversão de uma lista com e sem acumuladores.

```
inverte([],[]).
inverte([H|T],L) :- inverte(T,T1), concatena(T1,[H],L).
```

acumulador

```
inv(Xs,Ys) :- inv(Xs,[],Ys).

inv([X|Xs],Ac,Ys) :- inv(Xs,[X|Ac],Ys).
inv([],Ys,Ys).
```

Exercício:

1. Considere o seguinte procedimento para o cálculo do factorial

```
fact(0,1).
fact(N,F) :- N>0, N1 is N-1, fact(N1,F1), F is N*F1.
```

Defina uma outra versão de factorial que utilize um parâmetro de acumulação.

2. Defina uma versão do predicado somatório que utilize um acumulador.

Predicados de tipo *(meta-lógica)*

O Sicstus Prolog tem um conjunto de predicados pré-definidos que permitem fazer uma análise dos termos (ver *User's Manual*).

var nonvar atom number atomic simple compound ground integer float ...

```
functor(+Term,?Name,?Arity)
functor(?Term,+Name,+Arity)

arg(+ArgNo,+Term,?Arg)

+Term =.. ?List
?Term =.. +List

name(+Const,?CharList)
name(?Const,+CharList)
```

Exemplo:

```
| ?- functor(data(X,abril,2006),N,A).
A = 3,
N = data ?
yes
| ?- functor(X,hora,2).
X = hora(_A,_B) ?
yes
| ?- name('ABC',X), name(abc,Y), name(123,Z).
X = [65,66,67],
Y = [97,98,99],
Z = [49,50,51] ?
yes
```

```
| ?- arg(2,data(X,abril,2006),A).
A = abril ?
yes
| ?- data(X,abril,2006) =.. L.
L = [data,X,abril,2006] ?
yes
| ?- Z =.. [hora,12,30].
Z = hora(12,30) ?
yes
```


Exemplos: Duas implementações de predicados que testam a relação de subtermo.

```
% subtermo(T1,T2) testa se T1 é subtermo de T2
subtermo(T1,T2) :- T1 == T2.
subtermo(S,T) :- compound(T), functor(T,F,N), subtermo(N,S,T).

subtermo(N,S,T) :- N>1, N1 is N-1, subtermo(N1,S,T).
subtermo(N,S,T) :- arg(N,T,A), subtermo(S,A).
```

```
% subterm(T1,T2) testa se T1 é subtermo de T2
subterm(T,T).
subterm(S,T) :- compound(T), T =.. [F|As], subtermList(S,As).

subtermList(S,[A|R]) :- subterm(S,A).
subtermList(S,[A|R]) :- subtermList(S,R).
```

Note as diferenças entre as duas versões.

```
| ?- subterm(f(X),g(h(t,f(X)),a)).
true ?
yes
```

```
| ?- subtermo(f(t),g(h(t,f(X)),a)).
no
| ?- subterm(f(t),g(h(t,f(X)),a)).
X = t ?
yes
```