3. Assuma que para implementar uma agenda electrónica temos na base de conhecimento factos com a seguinte informação agenda(*data, horaInicio, horaFim, tarefa, tipo*). Por exemplo:

```
agenda(data(5,abril,2006), 9, 13, join, palestras).
agenda(data(6,abril,2006), 11, 13, join, palestras).
agenda(data(6,abril,2006), 16, 17, logcomp, aulas).
agenda(data(6,abril,2006), 17, 20, atendimento, aulas).
agenda(data(4,abril,2006), 15, 17, di, reuniao).
agenda(data(7,abril,2006), 8, 13, logcomp, aulas).
agenda(data(7,abril,2006), 15, 17, ccc, reuniao).
agenda(data(4,maio,2006), 11, 13, pure, seminarios).
```

a) Defina o predicado `cria_tipo(+Tipo)` que consulta a agenda e gera factos do nome do tipo com a lista de pares *(data,tarefa)* associados a esse tipo.

b) Defina o predicado `apaga_mes(+Mes)` que apaga todas as marcações de um dado mês.

c) Defina o predicado `marca(+Tarefa,+Tipo,+LDatas)` faz a marcação de uma dada tarefa, de um dado tipo, para uma lista de datas.

Input / Output

Para além dos canais de comunicação usuais com o utilizador (teclado/écran) um programa Prolog pode fazer input/output de informação através de outros canais, por exemplo: ficheiros.

Os canais de entrada de dados chamam-se ***input streams*** e os de saída ***output streams***.

O Sicstus Prolog tem diversos predicados pré-definidos para input/output de caracteres e de termos (ver *User's Manual*). A maioria destes predicados tem duas versões: *com* e *sem* a indicação explícita do *stream*. Quando o *stream* não é indicado, o *input* é do canal de entrada actual *(current input stream)* e o *output* é para o canal de saida actual *(current output stream)*.

No início da execução de um programa, os canais actuais de entrada e de saida são o teclado e o écran, e o nome destes canais é **user**. Mas o input e o output pode ser redireccionado através da invocação de predicados apropriados (como veremos).

Input / Output de Caracteres

O Sicstus Prolog tem diversos predicados pré-definidos para input/output de caracteres (ver *User's Manual*):

get_char get_code put_char put_code nl read_line skip_line ...

Exemplos:

```
| ?- get_char(X).
|: b

X = b ? yes
| ?- get_char(X).
|: K

X = 'K' ? yes
| ?- get_char(user,X).
|: 2

X = '2' ? yes
```

```
| ?- get_code(X).
|: A

X = 65 ? yes
| ?- get_code(X).
|: 1

X = 49 ? yes
| ?-
get_code(user,X).
|: z

X = 122 ? yes
```

```
| ?- get_char(a).
|: a

yes
| ?- get_char(a).
|: b

no
| ?- get_code(65).
|: A

yes
```


Exemplos:

```
| ?- put_char(X).
! Instantiation error in argument 1 of put_char/1
! goal:  put_char(_73)
| ?- put_char('X').
X
yes
| ?- put_char(a).
a
yes
| ?- put_char(1).
! Type error in argument 1 of put_char/1
! character expected, but 1 found
! goal:  put_char(1)
| ?- put_char('1').
1
yes
| ?- put_char(user,'a').
a
yes
```

```
| ?- read_line(X).
|: A a 1 Bb
X = [65,32,97,32,49,32,66,98] ?
yes
```

```
| ?- skip_line.
|: isto serve de exemplo
yes
| ?- nl.

yes
```

Exemplos:

```
| ?- put_code(65).
A
yes
| ?- put_code(a).
! Type error in argument 1 of put_code/1
! integer expected, but a found
! goal:  put_code(a)
| ?- put_code(X).
! Instantiation error in argument 1 of put_code/1
! goal:  put_code(_73)
| ?- put_code(32).

yes
| ?- put_code(user,120).
x
yes
```

Existem outros predicados pré-definidos para input/output de caractes. Em particurar, os tradicionalmente usados (embora tendam a cair em desuso):

get, **get0** e **put** são semelhantes a **get_char**, **get_code** e **put_code**, respectivamente.

skip e **ttynl** são semelhantes a **skip_line** e **nl**.
tab(*+N*) escreve *N* espaços; etc ...

Input / Output de Termos

O Sicstus Prolog tem diversos predicados pré-definidos para input/output de termos (ver *User's Manual*):

read(*?Term*) — lê o próximo termo da *stream* de entrada e unifica-o com *Term*. Se a *stream* já tiver chegado ao fim *Term* unifica com o átomo **end_of_file**. Note que cada termo tem que ser seguido de ponto final e *enter* ou espaço.

write(*?Term*) — escreve o termo *Term* na *stream* de saída.

writeq(*?Term*) — escreve o termo *Term* na *stream* de saída, e coloca pelicas sempre que isso seja necessário (para que esse termo possa, posteriormente, ser lido pelo predicado `read`).

Exemplo:

```
| ?- write('Escreva um número: '), read(N), nl, N1 is N*N,
     write('O quadrado de '), write(N),  write(' é '), write(N1), nl, nl.
Escreva um número: 3.

O quadrado de 3 é 9

N = 3,
N1 = 9 ?
yes
```


`read(T)` faz com que o próximo termo da *input stream* unifique com T. Se a unificação não for possível, o predicado falha e não há backtracking para ler outro termo.

Exemplos:

```
| ?- read(X).
|: um exemplo.
! Syntax error in read/1
! ...
| ?- read(X).
|: 'um exemplo'.
X = 'um exemplo' ?
yes
| ?- read(user,X).
|: amanha.
X = amanha ?
yes
| ?- read(X).
|: 'Amanha'.
X = 'Amanha' ?
yes
| ?- read(X).
|: 'um exemplo
     com mais de uma linha'.
X = 'um exemplo\ncom mais de uma linha' ?
yes
```

```
| ?- read(X).
|: [1,2,3,4].
X = [1,2,3,4] ?
yes
| ?- read(user,Z).
|: 1+3*5.
Z = 1+3*5 ?
yes
| ?- read(a).
|: a.
yes
| ?- read(a).
|: b.
no
| ?- read(1+X).
|: 1+3*5.
X = 3*5 ?
yes
| ?- read(X*5).
|: 1+3*5.
no
```


`writeq(T)` coloca pelicas em T sempre que necessário. Um termo escito com `writeq` pode depois ser sempre lido com `read`.

Exemplos:

```
| ?- read(X), write(X), nl, writeq(X).
|: 'bom dia'.
bom dia
'bom dia'
X = 'bom dia' ?
yes
| ?- read(X), write(X), nl, writeq(X).
|: olá.
olá
olá
X = olá ?
yes
| ?- read(X), write(X), nl, writeq(X).
|: Olá.
_430
_430
true ?
yes
| ?- read(X), write(X), nl, writeq(X).
|: 'Olá'.
Olá
'Olá'
X = 'Olá' ?
yes
```

Exemplos:

```
| ?- read(data(D,M,A)), write(D), write(' de '), write(M),
     write(' de '), write(A).
|: data(15,abril,2006).
15 de abril de 2006
A = 2006, D = 15, M = abril ?
yes
| ?- read(data(D,M,A)), write(D), write(' de '), name(M,[H|T]),
     H1 is H-32, name(M1,[H1|T]), write(M1),  write(' de '), write(A).
|: data(15,abril,2006).
15 de Abril de 2006
A = 2006, D = 15, H = 97, M = abril, T = [98,114,105,108], H1 = 65, M1 = 'Abril' ?
yes
| ?- read(X), X =.. [horas,H,M], write(H), write(:), write(M).
|: horas(10,25).
10:25
H = 10, M = 25, X = horas(10,25) ?
yes
```

O Sicstus Prolog tem muitos outros predicados de IO, como por exemplo o predicado `format` que tem semellhanças com o *printf do C* (ver *User's Manual*).

`format`(*+Format*, *:Arguments*) escreve no output stream de acordo com *Format* e a lista de argumentos *Arguments*.

Exemplos:

```
| ?- read(horas(H,M)), format('São ~d horas e ~d minutos!',[H,M]).
|: horas(9,45).
São 9 horas e 45 minutos!
H = 9,  M = 45 ?
yes
| ?- read(X), X =.. L, format('São ~i~d horas e ~d minutos!',L).
|: time(15,30).
São 15 horas e 30 minutos!
L = [time,15,30],  X = time(15,30) ?
yes
| ?- format('Escreva um número: ',[]), read(N), Q is N*N,
     format('O quadrado de ~2f é ~2f \n\n',[N,Q]).
Escreva um número: 2.3.
O quadrado de 2.30 é 5.29

N = 2.3,  Q = 5.289999999999999 ?
yes
| ?- format('Olá, ~a ~s !', [muito,"bom dia"]).
Olá, muito bom dia !
yes
| ?- format('Aqui ~a um ~20s!!!', ['está mais',"bom exemplo"]).
Aqui está mais um bom exemplo         !!!
yes
```



```
conversoes :- write('Escreva uma lista de números (ou fim para terminar):'),
              nl, read(L), processa(L).

processa(fim) :- !.
processa(L) :- format('\nCelcius \tFarenheit \tKelvin\n',[]),
               format('~40c\n',[0'-]), converte(L).

converte([]) :- nl, conversoes.
converte([C|T]) :- F is C * 1.8 + 32, K is C + 273,
                   format('~2f \t\t~2f \t\t~2f\n',[C,F,K]), converte(T).
```

Exemplo:

Este programa lê uma lista de temperaturas em graus Celcius e gera uma tabela com o resultado das conversões.

```
| ?- conversoes.
Escreva uma lista de números (ou fim para terminar):
|: [27.4,32.75,100].

Celcius         Farenheit       Kelvin
----------------------------------------
27.40           81.32           300.40
32.75           90.95           305.75
100.00          212.00          373.00

Escreva uma lista de números (ou fim para terminar):
|: [0, -15.4, -28.2, 12.7].

Celcius         Farenheit       Kelvin
----------------------------------------
0.00            32.00           273.00
-15.40          4.28            257.60
-28.20          -18.76          244.80
12.70           54.86           285.70

Escreva uma lista de números (ou fim para terminar):
|: fim.
yes
```


Exercícios:

1. Defina o predicado `tabuada(+N)` que dado um número inteiro `N`, apresenta no écran a tabuada do `N`.

2. Escreva um programa `escreve_tabuadas` que lê um inteiro do teclado, escreve no écran a sua tabuada, e continua pronto para escrever tabuadas até que seja mandado terminar.

3. Escreva um programa que lê uma lista de pares *(átomo, nº inteiro)* e apresenta um gráfico de barras dessa lista de pares. (Cada unidade deve ser representada pelo caracter #, e as barras podem ser horizontais.)

4. Escreva um programa que lê os coeficientes de um polinómio de 2º grau $ax^2 + bx + c$ e calcula as raizes reais do polinómio, apresentando-as no écran. Se o polinómio não tiver raizes reais, o programa deve informar o utilizardor desse facto.

Ficheiros

Os ficheiros são vistos como *streams*. A *stream* corresponde ao descritor do ficheiro (ao nível do sistema operativo.)

Um ficheiro pode ser aberto para leitura ou escrita através do predicado **open**. Este predicado devolve o descritor do ficheiro que pode depois ser passado como argumento dos predicados de I/O. Para fechar o ficheiro usa-se o predicado **close**.

Existem predicados para saber informação sobre as streams existentes.

O Sicstus Prolog tem um vasto conjunto e predicados para manilupação de streams (ver *User's Manual*). Ficam aqui alguns exemplos:

open(*+FileName*,*+Mode*,*-Stream*) abre o ficheiro *FileName* em modo *Mode* (pode ser: read, write ou append). *Stream* fica como descritor desse ficheiro.

set_input(*+Stream*) torna *Stream* o canal actual de entrada.
set_output(*+Stream*) torna *Stream* o canal actual de saída.
current_input(*?Stream*) *Stream* é o canal actual de entrada.
current_output(*?Stream*) *Stream* é o canal actual de saída.
current_stream(*?FileName*,*?Mode*,*?Stream*) serve para saber informação sobre as *Streams*.

flush_output(*+Stream*) descarrega o *buffer* do canal *Stream*.
close(*+Stream*) fecha o canal *Stream*.

Exemplo:

```
iniciaConv :- open('CelFar.txt',append,CF),
              open('CelKel.txt',write,CK), convFich(CF,CK).

convFich(CF,CK) :- write('Escreva uma lista de números (ou fim.):'),
                   nl, read(user,L), trata(L,CF,CK).

trata(fim,CF,CK) :- close(CF), close(CK), !.
trata(L,CF,CK) :- format(CF,'\nCelcius \tFarenheit\n~25c\n',[0'-]),
                  format(CK,'\nCelcius \tKelvin\n~22c\n',[0'-]),
                  converteF(L,CF,CK).

converteF([],CF,CK) :- nl(user), convFich(CF,CK).
converteF([C|T],CF,CK) :- F is C*1.8+32, format(CF,'~2f \t\t~2f\n',[C,F]),
                          K is C+273, format(CK,'~2f \t\t~2f\n',[C,K]),
                          converteF(T,CF,CK).
```

Note que o ficheiro *CelFar.txt* é aberto em modo append, enquanto o ficheiro *CelKel.txt* é aberto em modo write.


```
| ?- iniciaConv.
Escreva uma lista de números (ou fim.):
|: [27.4,32.75,100].

Escreva uma lista de números (ou fim.):
|: fim.
yes

| ?- iniciaConv.
Escreva uma lista de números (ou fim.):
|: [55, 23.6].

Escreva uma lista de números (ou fim.):
|: [-10.8, 0, 20.2].

Escreva uma lista de números (ou fim.):
|: fim.
yes
```

CelFar.txt

```
Celcius          Farenheit
------------------------
27.40            81.32
32.75            90.95
100.00           212.00

Celcius          Farenheit
------------------------
55.00            131.00
23.60            74.48

Celcius          Farenheit
------------------------
-10.80           12.56
0.00             32.00
20.20            68.36
```

CelKel.txt

```
Celcius          Kelvin
----------------------
55.00            328.00
23.60            296.60

Celcius          Kelvin
----------------------
-10.80           262.20
0.00             273.00
20.20            293.20
```


tell, telling, told ... see, seeing, seen

São os predicados Prolog tradicionalmente usados na manipulação de ficheiros.

tell(*+File*) abre *File* para escrita e torna-o canal de saída actual. Se *File* já tiver aberta apenas o torna canal de saída actual.

telling(*?FileName*) unifica *FileName* com o nome do ficheiro de saída actual se este foi aberto com tell, senão unifica com user.

told fecha o canal actual de saída (e este volta a ser user).

see(*+File*) abre *File* para leitura e torna-o canal de entrada actual. Se *File* já tiver aberta apenas o torna canal de entrada actual.

seeing(*?FileName*) unifica *FileName* com o nome do ficheiro de entrada actual se este foi aberto com see, senão unifica com user.

seen fecha o canal de entrada actual (e este volta a ser user).

Exemplos:

```
| ?- see('teste.txt'), read(T), seeing(X), seen.
! Existence error in argument 1 of see/1
! file 'teste.txt' does not exist
! goal:  see('teste.txt')
| ?- tell('teste.txt'),
     writeq('Isto é uma experiencia!'), write('.'), nl,
     telling(X), told.
X = 'teste.txt' ?
yes
| ?- see('teste.txt'), read(T1), read(T2), seeing(X), seen.
X = 'teste.txt',
T1 = 'Isto é uma experiencia!',
T2 = end_of_file ?
yes
| ?- seeing(X), telling(Y).
X = user, Y = user ?
yes
```


Exemplos:

```
| ?- tell(aaa), format('Uma ~a experiencia!',[nova]), telling(X), told.
X = aaa ?
yes
| ?- see(aaa), get_char(C1), get_char(C2).
C1 = 'U',
C2 = m ?
yes
| ?- get_char(C3), seeing(X).
X = aaa,
C3 = a ?
yes
| ?- seen, get_char(C4).
|: z

C4 = z ?
yes
```

```
| ?-  open(aaa,read,A), seeing(X), get_char(C), get_char(A,K), close(A).
|: z

A = '$stream'(4085968),
C = z,
K = 'U',
X = user ?
yes
```


Os predicados `telling` e `seeing` podem ser usados para recolher informação sobre os canais de saída e de entrada (num dado momento) de forma a mais tarde se conseguir repôr o mesmo contexto de comunicação. Usam-se as combinações:

```
telling(F), tell(file), ... , told, tell(F)
```

```
seeing(F), see(file), ... , seen, see(F)
```

Exemplo:

```
| ?- tell(fam), write('pai(rui,carlos).'), nl.
yes
| ?- write(pai(pedro,hugo)), write('.'), nl.
yes
| ?- telling(F), tell(ami),
     write('amigos(ana,rui).'), nl, write('amigos(hugo,ana).'), nl,
     told, tell(F).
F = fam ?
yes
| ?- write('pai(paulo,ricardo).'), nl, write('pai(manuel,helena).'), nl.
yes
| ?- telling(X), told.
X = fam ?
yes
```

Produz os ficheiros

fam

```
pai(rui,carlos).
pai(pedro,hugo).
pai(paulo,ricardo).
pai(manuel,helena).
```

ami

```
amigos(ana,rui).
amigos(hugo,ana).
```


"Repeat loops"

O Sicstus Prolog tem pré-definido o predicado **repeat**, que é um predicado de controlo que permite fazer o controlo do backtracking, e que deve ser usado em combinação com o **cut** para gerar ciclos.

O esquema geral de programação com *repeat loops* é o seguinte:

```
Head :- ...,
        repeat,
          generate(Datum),
          action(Datum),
          test(Datum),
        !,
        ...
```

O seu propósito é repetir uma determinada *acção* sobre os elementos que vão sendo *gerados* (por exemplo, que vão sendo lidos de uma *stream*) até que uma determinada condição de *teste* seja verdadeira. Note que estes ciclos só têm interesse se envolverem efeitos laterais.

A utilização típica de *repeat loops* é no processamento de informação lida de ficheiros e na interacção com o utilizador (na implementação de menus e na validação dos dados de entrada).