Le professeur de la classe de mathématiques a perdu le classement de ses onze élèves et ne dispose, pour le reconstituer, que des indications que veut bien lui donner l'élève Lolo :

1. Le premier et le dernier de la liste portent des noms ayant la même initiale.
2. Aucun élève n'a la même initiale que son ou ses voisins de classement.
3. William et Wilfrid sont après Washington qui, au classement, n'est pas voisin de Ludwig.
4. Herbert, par contre, est au classement, voisin de Ludwig.
5. Ludovic, voisin de Wolfgang au classement, occupe l'une des trois dernières places.
6. Le nombre de places séparant Hubert de Ludwig est égal au nombre de places séparant Hubert de William, lequel est à moins de trois places d'Honoré.
7. Il n'y a pas d'ex-æquo et Wilfrid est avant moi.
8. Un seul d'entre nous occupe la place qu'il aurait dans l'ordre alphabétique.
9. Harold occupe une place impaire et est mieux classé que William.



```
neighbour(X,Y,Zs) :- append(Xs,[X,Y|Ys],Zs).
neighbour(Y,X,Zs) :- append(Xs,[X,Y|Ys],Zs).

not_neighbour(X,Y,Zs) :-
   append(Xs,[A|Ys],Zs),
   member(X,Xs), member(Y,Ys).
not_neighbour(Y,X,Zs) :-
   append(Xs,[A|Ys],Zs),
   member(X,Xs), member(Y,Ys).

before(X,Y,Zs) :- append(Xs,[Y|Ys],Zs),
                  member(X,Xs).

odd_rank(X,[X|Xs]).
odd_rank(X,[A,B|Xs]) :- odd_rank(X,Xs).
```



```
less_than_three(X,Y,St) :-
   append(Xs,[X,Y|Ys],St).
less_than_three(Y,X,St) :-
   append(Xs,[X,Y|Ys],St).
less_than_three(X,Y,St) :-
   append(Xs,[X,A,Y|Ys],St).
less_than_three(Y,X,St) :-
   append(Xs,[X,A,Y|Ys],St).
```

Le prédicat `up_mid` permet de déterminer si trois élèves `A`, `M` et `B` se succèdent dans cet ordre au sein du classement `St`, avec la restriction additionnelle que l'écart entre `A` et `M` est le même que celui entre `M` et `B`.

```
up_mid(A,M,B,St) :-
 append(Xs,[A,M,B|Ys],St).
up_mid(A,M,B,St) :-
 append(Xs,[A,A1,M,B1,B|Ys],St).
up_mid(A,M,B,St) :-
 append(Xs,[A,A1,A2,M,B2,B1,B|Ys],St).
up_mid(A,M,B,St) :-
 append(Xs,
        [A,A1,A2,A3,M,B3,B2,B1,B|Ys],St).
up_mid(A,M,B,
       [A,A1,A2,A3,A4,M,B4,B3,B2,B1,B]).
```

Le prédicat `middle` permet de détecter la situation analogue dans le même ordre ou dans l'ordre inverse.

```
middle(A,M,B,St) :- up_mid(A,M,B,St).
middle(B,M,A,St) :- up_mid(A,M,B,St).
```

/* deux variantes */

```
odd_rank(X1, [X1,X2,...,X11]).
odd_rank(X3, [X1,X2,...,X11]).
...
odd_rank(X11,[X1,X2,...,X11]).
```

C'est peu concis, mais simple et efficace.

```
...
odd_rank(X3,[_,_,X3,_,_,_,_,_,_,_,_]).
...
```

"_" est un nom générique pour une variable "anonyme".

```
alpha([harold,herbert,honore,hubert,
       lolo,ludovic,ludwig,
       washington,wilfrid,
       william,wolfgang]).

init(X,h) :-
 member(X,[harold,herbert,honore,hubert]).
init(X,l) :-
 member(X,[lolo,ludovic,ludwig]).
init(X,w) :-
 member(X,[washington,wilfrid,
           william,wolfgang]).

exactly_one([X|Xs],[X|Ys]) :-
 exactly_none(Xs,Ys).
exactly_one([X|Xs],[Y|Ys]) :-
 X \= Y, exactly_one(Xs,Ys).

exactly_none([],[]).
exactly_none([X|Xs],[Y|Ys]) :-
 X \= Y, exactly_none(Xs,Ys).
```



```
distinct(h,l). distinct(h,w).
distinct(l,h). distinct(l,w).
distinct(w,h). distinct(w,l).

distinct_init_neighbour([X]).
distinct_init_neighbour([X,Y|Ys]) :-
      init(X,I), init(Y,J), I \= J,
      distinct_init_neighbour([Y|Ys]).
```



Prédicat question.

```
go(St) :-   /* indices, dans l'ordre  */
  St = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11],
  init(X1,I), init(X11,I),
  distinct_init_neighbour(St),
  before(washington,william,St),
  before(washington,wilfrid,St),
  not_neighbour(washington,ludwig,St),
  neighbour(herbert,ludwig,St),
  neighbour(ludovic,wolfgang,St),
  member(ludovic,[X9,X10,X11]),
  middle(ludwig,hubert,william,St),
  less_than_three(william,honore,St),
  before(wilfrid,lolo,St),
  alpha(L), exactly_one(St,L),
  odd_rank(harold,St),
  before(harold,william,St).
```

Ordre inadéquat.

La question `go(St)` génère quantité d'essais infructueux ; par exemple, le "classement" :

```
[harold, lolo, harold, ludwig,
 washington, ludwig, william,
 ludwig, herbert, wilfrid, harold]
```

quoique grossièrement impossible (certains élèves n'apparaissent pas, d'autres apparaissent deux fois ou plus), respecte quand même les quatre premiers indices.

Ordre adéquat

```
go(St) :-
  St = [X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11],
  member(ludovic,[X9,X10,X11]),
  odd_rank(harold,St),
  before(harold,william,St),
  neighbour(ludovic,wolfgang,St),
  neighbour(herbert,ludwig,St),
  before(wilfrid,lolo,St),
  before(washington,william,St),
  not_neighbour(washington,ludwig,St),
  less_than_three(william,honore,St),
  before(washington,wilfrid,St),
  middle(ludwig,hubert,william,St),
  alpha(L), exactly_one(St,L),
  init(X1,I), init(X11,I),
  distinct_init_neighbour(St).
```



Le système fournit alors l'unique solution au problème ; le classement correct est :

```
[washington, herbert, ludwig,
 wilfrid, harold, lolo, hubert,
 wolfgang, ludovic, honore, william] ;
```

Le prédicat `not_neighbour` est vrai si et seulement si le prédicat `neighbour` est faux.

On peut remplacer le prédicat

```
not_neighbour(washington,ludwig,St)
```

par le prédicat

```
not(neighbour(washington,ludwig,St))
```