1) Considérons la représentation structurelle d'un circuit combinatoire, sous la forme d'un ensemble de portes logiques.

Un circuit logique combinatoire est représenté par un terme de forme `circuit(Gates)`, où `Gates` est une liste de portes logiques ou de commentaires. Chaque porte est éventuellement précédée d'un commentaire. Une porte logique est un terme de la forme `gate(Name, Type, Inputs, Output)`. `Name` est un nom unique, associé à la porte respective. `Type` est le type de la porte et il peut être `or`, `and`, `xor`, `not`, `nor` ou `nand`. `Inputs` représente les entrées de la porte respective. Pour les portes `not`, il n'y a qu'une seule entrée, qui est représentée par un symbole. Pour les autres portes, `Inputs` est une liste de symboles. `Output` est un symbole qui montre la sortie de la porte.

Par exemple, un additionneur complet à un bit est représenté par:

```
circuit([
gate(x1, xor, [i1, i2], t1),
com('Gate to generate the sum bit'),
gate(x2, xor, [t1, i3], o1),
gate(a1, and, [i1, i2], t2),
gate(a2, and, [i3, t1], t3),
com('Gate to generate the transport bit'),
gate(o1, or, [t3, t2], o2)]).
```

Exigences:

a) Définir le prédicat `circuit(X)`, qui est vrai si `X` est un terme, qui représente un circuit de logique combinatoire, sous la forme d'une liste de circuits.

b) Définir le prédicat `delete_comments(Ci, Ce)`, qui est vrai si `Ce` est la description du circuit obtenue en éliminant les commentaires de la description du circuit `Ci`.

c) Définir le prédicat `connexions_list(C, Connections)`, qui est vrai si `Connections` est la liste de tous les noms de connexions du circuit `C`.

d) Supposons que les valeurs des signaux d'entrée du circuit soient données sous la forme d'une liste, `Signals`, avec des éléments de la forme `signal(Input, Value)`. Par exemple, si l'additionneur complet reçoit en entrée les valeurs `i1=0`,

`i2=1, i3=1`, alors cette liste sera `[signal(i1, 0), signal(i2, 1), signal(i3, 1)]`. Définir le prédicat `signal(Connection, Circuit, Inputs, Value)`, qui est vrai si `Value` est la valeur de la connexion `Connection` du circuit `Circuit`, pour les entrées avec les valeurs de la liste `Inputs`.

2) Considérons le problème des 15-puzzle, pour une grille carrée de taille 4 * 4. La grille contient 15 carrés glissants. Chaque carré est étiqueté avec un nombre naturel compris entre 1 et 15. Il y a un carré vide qui permet le glissement des carrés, avec une position à gauche, à droite, en haut ou en bas.
Il est nécessaire de trouver une séquence minimale de déplacements qui transforme une configuration initiale dans une configuration finale donnée. Le problème peut être résolu en utilisant une stratégie de recherche dans l'espace d'états.
Un état du problème est donné par la configuration des carrés sur la grille. Cette configuration peut être représentée par un terme `st(List, Position)`. `List` est une liste de 16 nombres naturels. Les nombres représentent les étiquettes des carrés, allant sur les lignes, de gauche à droite. Le carré vide est considéré comme étiqueté avec la valeur 0. `Position` représente l'index de la position vide dans cette liste (les index sont comptés à partir de 1).
Par exemple, le tableau suivant:

1	3	6	9
10	2	13	15
14	7		5
8	4	11	12

est représenté par la liste Prolog:

```
st([1,3,6,9,10,2,13,15,14,7,0,5,8,4,11,12],11).
```

Exigences:
a) Définir le prédicat `state(X)`, qui est vrai si `X` est un terme, qui représente un état du problème des 15-puzzle.
b) Définir le prédicat `neighbours(S0, LS1)`, qui est vrai si `LS1` est la liste des prochains états, suivant l'état `S0`, dans le problème des 15-puzzle. Les mouvements sont considérés dans l'ordre suivant: gauche, haut, droite, bas.

c) Pour résoudre le problème des 15-puzzle à l'aide d'un algorithme de recherche heuristique, il est nécessaire d'utiliser une fonction heuristique, dont la valeur représente une estimation du nombre de mouvements nécessaires pour passer de l'état actuel à l'état final. Soit `S0` et `S1` deux états. La distance qui les sépare peut être estimée en comptant toutes les paires de carrés placés dans des positions similaires dans les deux états, mais étiquetés avec des valeurs différentes. Définir le prédicat `eval_h(S0, S1, H)`, qui est vrai si `H` est la valeur de l'estimation heuristique de la distance entre les états `S0` et `S1` du problème des 15-puzzle.

d) Définir le prédicat `neighbours2(S0,LS1)`, qui est vrai si `LS1` est la liste des états successeurs, suivant l'état `S0`, obtenu en effectuant deux mouvements. Les mouvements sont considérés dans l'ordre suivant: gauche, haut, droite, bas. Revenir dans `S0` sera ignoré.

3) Résoudre une version du problème du singe et de la banane. Pour plus de détails sur ce problème, lisez le fichier joint Monkey.pdf et voir le film de https://www.youtube.com/watch?v=T095SaEhHas

Dans une chambre, il y a un singe, une boîte dans un coin (ou près de la fenêtre) et une banane, suspendue au plafond, au milieu de la chambre. Le singe veut manger la banane, mais il ne peut le faire que s'il se déplace vers la boîte, place la boîte sous l'endroit où la banane est suspendue, monte sur la boîte et saute pour obtenir la banane.

Résoudre la version du problème du singe et de la banane, qui est étendue de la manière suivante:

a) L'environnement dispose de 4 chambres, étiquetées 1, 2, 3, 4

b) Il y a des portes de la chambre 4 vers les chambres 1, 2 et 3

c) Pour entrer dans une chambre et en sortir, le singe doit ouvrir la porte

d) Chaque chambre a une fenêtre

e) Dans chaque chambre se trouve une boîte (étiqueté aussi 1, 2, 3, 4, selon la chambre dans laquelle se trouve la boîte), près de la fenêtre

f) La banane est placée dans la chambre 4, suspendue au plafond

g) Le singe peut être initialement dans n'importe quelle chambre, dans l'un des endroits suivants: près de la porte, au milieu de la chambre, près de la fenêtre

h) Les boîtes peuvent être superposées, dans la même chambre, mais dans un ordre croissant de leurs index (il est possible de mettre la boîte 3 au-dessus de la boîte 1, la boîte 4 au-dessus de la boîte 2, mais il n'est pas possible de mettre la boîte 3 au-dessus de la boîte 4)

i) Parce que la banane est suspendue à une hauteur élevée, le singe doit placer les boîtes 1, 2, 3 et 4 l'une sur l'autre (dans cet ordre) dans la chambre 4, pour atteindre la banane

Le programme doit afficher la solution trouvée, sous la forme d'une séquence d'états, dans laquelle le singe est placé au hasard dans l'une des chambres.