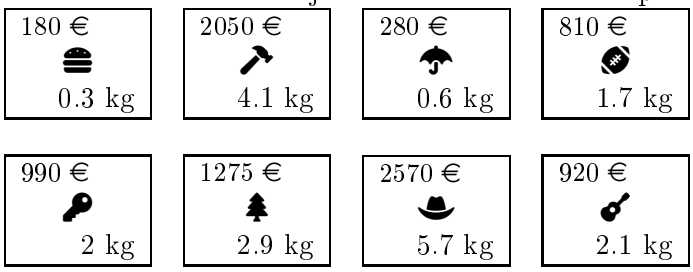
❑ Exercice 1 : *Problème du sac à dos*

On considère un problème du sac à dos avec les objets suivants et un sac de poids maximal 8kg.



1. En Ocaml, mettre en oeuvre une stratégie gloutonne afin de résoudre ce problème.
   ⊗ On pourra suivre les étapes suivantes :
   — Commencer par définir les objets, sous la forme d'une liste de triplets `string*int*float` (le hamburger est donc le triplet `("Hamburger",180,0.3)`.
   — Utiliser `List.sort` afin de classer les objets en utilisant un critère *pertinent* comme par exemple le rapport valeur/poids. On rappelle que `List.sort : ('a -> 'a -> int) -> 'a list -> 'a list` prend en premier paramètre une fonction de comparaison qui doit renvoyer un entier négatif si le premier argument est inférieur au second, 0 s'il est égal et un entier positif sinon.
   — Utiliser la stratégie gloutonne consistant à prendre les objets les mieux classés tant qu'il rentre dans le sac
2. En C, mettre en oeuvre une stratégie par force brute afin de résoudre ce problème.
   ⊗ On pourra suivre les étapes suivantes :
   — Définir un type structuré `objet` afin de représenter les objets
   — Représenter un choix d'objet par un tableau de booléens `c` tel que `c[i]` vaut `true` si l'objet `i` est choisi.
   — Ecrire une fonction de signature
   `void valeurpoids(objet objets[], bool choix[], int n, int *valeur, double *poids)`
   qui calcule la valeur et le poids d'une combinaison d'objet et modifie en conséquence les pointeurs passés en argument.
   — Enumérer les combinaisons possibles et prendre la combinaison ayant la plus grande valeur en respectant la contrainte de poids.

❑ Exercice 2 : *Code de Gray*

Afin d'énumérer les nombres à n chiffres en écriture binaire, on peut utiliser un *compteur binaire*. On démarre du nombre `0...0` (n chiffres) et à chaque étape, on incrémente ce nombre de 1. Afin de réaliser cet incrément, il suffit de rechercher le premier chiffre valant `0` (en partant de la droite), de le passer à `1` et de passer tous les chiffres à sa droite à `0`. Par exemple, pour $n = 4$ en soulignant le premier chiffre le plus à droite et valant `0` :

— `000`$\underline{\texttt{0}}$ devient `0001`
— `0`$\underline{\texttt{0}}$`11` devient `0100`

1. Ecrire une fonction `bool incremente(bool compteur[], int n)` qui prend en argument un tableau de booléens représentant un nombre en écriture binaire et incrémente ce compteur lorsque cela est possible et renvoie alors `true` et ne modifie pas le compteur et renvoie `false` sinon.
2. Quelle est la complexité en nombre de modifications de chiffres de la fonction `incremente` ?
3. Quelle est la complexité *amortie* d'une suite d'incrémentation sur un compteur binaire ?
   ⊗ Pour un compteur binaire C, noter $\Phi(C)$ le nombre de 1 dans C et montrer qu'il s'agit d'une fonction de potentiel.
4. Retrouver le résultat de la question précédente en calculant le nombre de fois où chacun des chiffres du compteur binaire est modifié lors de N incrémentations.

Les codes de Gray permettent d'énumérer toutes les valeurs prises par un compteur binaire en ne modifiant qu'un seul chiffre à chaque incrémentation. Par exemple, pour un compteur à 3 chiffres, les valeurs successives seront : `00`$\underline{\texttt{0}}$, `0`$\underline{\texttt{0}}$`1`, `01`$\underline{\texttt{1}}$, `0`$\underline{\texttt{1}}$`0`, `11`$\underline{\texttt{0}}$, `1`$\underline{\texttt{1}}$`1`, `10`$\underline{\texttt{1}}$, `100` (on a souligné à chaque étape l'unique chiffre qui sera modifié). Afin d'énumérer les codes de Gray successifs, une méthode consiste à compter le nombre de 1 dans le compteur :

1. si ce nombre est pair alors on inverse le dernier chiffre,
2. sinon on inverse le chiffre situé à gauche du 1 le plus à droite.

On propose le type structuré suivant afin de représenter un code de Gray :

```c
struct gray_s
{
    bool *compteur;
    bool parite;
    int taille;
};
typedef struct gray_s gray;
```

5. Ecrire une fonction de signature `gray cree(int n)` qui renvoie un code de gray à n chiffres.
6. Ecrire une fonction de signature `int dernier_un(gray g)` qui renvoie l'indice du dernier 1 dans un code de Gray.
7. Ecrire une fonction de signature `bool ajoute(gray *g)` qui incrémente un code de Gray.

❑ **Exercice 3** : *Recherche de carrés magiques en retour sur trace*

Pour un entier n quelconque, le but de l'exercice est de programmer en C une recherche en *backtracking* d'une solution au problème du placement des entiers $1, \dots, n^2$ entiers dans un carré de côté n afin de former un carré magique.

1. Vérifier que dans le cas $n = 3$ le carré suivant est une solution :

2	7	6
9	5	1
4	3	8

2. Quelle serait la somme de chacune des lignes, colonnes ou diagonales dans le cas $n = 4$?
3. Donner l'expression de la somme de chacune des lignes, colonnes, ou diagonales pour un entier n quelconque.

Afin d'implémenter la résolution en C, on propose de linéariser le carré en le représentant par un tableau à une seule dimension. Le carré ci-dessus est par exemple représenté par :
`int carre[9] = {2, 7, 6, 9, 5, 1, 4, 3, 8}`

4. Donner l'expression de l'indice d'un élément situé en ligne i, colonne j dans le tableau linéarisé.
5. Inversement, donner la ligne et la colonne dans le carré initial d'un élément situé à l'indice i dans le tableau linéarisé.

On représente par l'entier 0 une case non encore rempli du tableau, par exemple :
`int carre[9] = {1, 8, 4, 0, 0, 0, 0, 0, 0}` représente un carré ayant simplement la première ligne complète

6. Ecrire une fonction `bool valide_ligne(int carre[], int lig, int size, int somme)` qui vérifie que la ligne `lig` d'un carré en cours de construction est encore valide, c'est le cas si cette ligne contient un zéro (car elle est alors incomplète) ou si elle ne contient aucun zéro et que sa somme est égale à la variable `somme` fournie en argument.

On suppose écrite les fonctions correspondantes pour les colonnes et les lignes de même qu'une fonction : `bool valide_carre(int carre[], int size, int somme)` qui vérife qu'un carré en cours de construction est encore valide.

7. Ecrire la fonction permettant de rechercher par backtracking un carré magique solution du problème posé. On pourra passer en argument à cette fonction un tableau de booléens de taille n^2 dont l'élément d'indice i indique si l'entier i a déjà été placé ou non dans le carré.