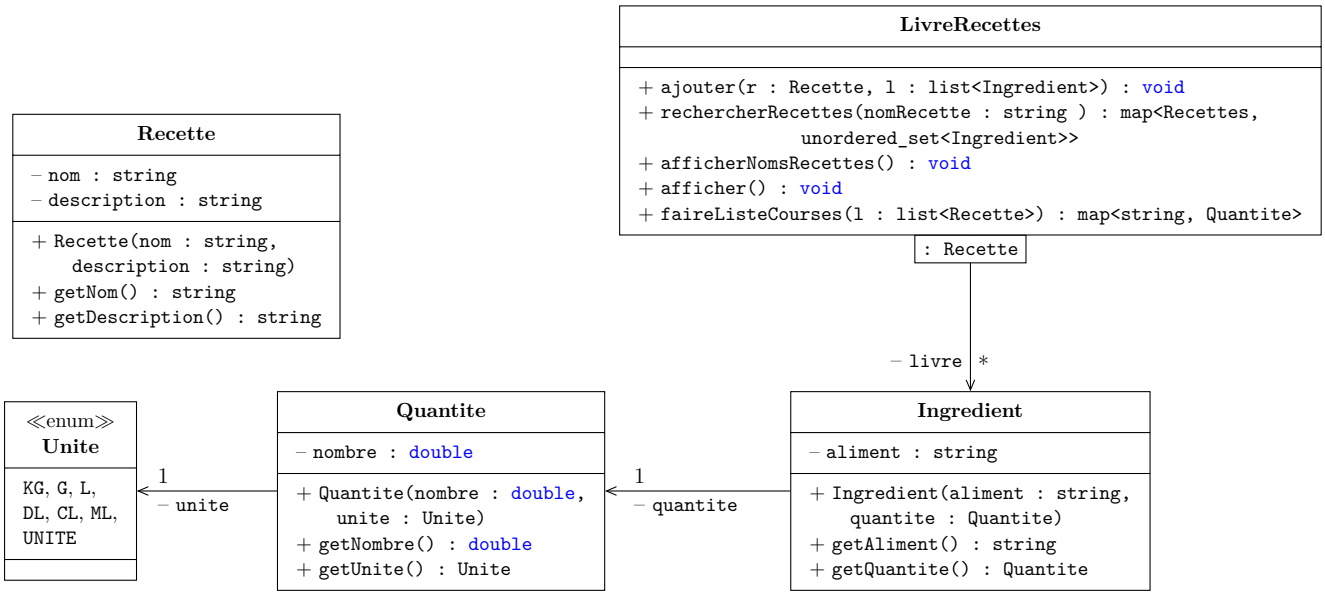
TP5
Livre de recettes de cuisine

Ce TP comportera un namespace `appli` qui va être détaillé dans la suite de ce sujet. Téléchargez le `main.cpp` qui est sur l'ENT et complétez le code.

1 Namespace `appli`



Codez les classes `Unite`, `Quantite`, `Ingredient` et `Recette` comme indiqué sur le diagramme ci-dessus.

Attention ! Ne codez pas la classe `LivreRecettes` pour l'instant !

Vous devrez respecter les indications suivantes :

— L'énumération `Unite` comprend les constantes : `G`, `KG`, `ML`, `CL`, `DL`, `L` et `UNITE`. Lorsque l'on affiche une `Unite`, on veut que soit affiché respectivement : `g`, `kg`, `mL`, `cL`, `dL`, `L` et rien. Il faut donc redéfinir l'opérateur `<<` ainsi.

— Une quantité permet de représenter `1,5kg`, `20cL`, ou `1 UNITE` (autrement dit, `1`) par exemple.

— Par défaut, le constructeur de `Quantité` met le nombre à 1 et l'unité à `UNITE`.

— Un ingrédient est ici un aliment pris en une certaine quantité. Par exemple : `3 poireaux` ou `1dL lait`.

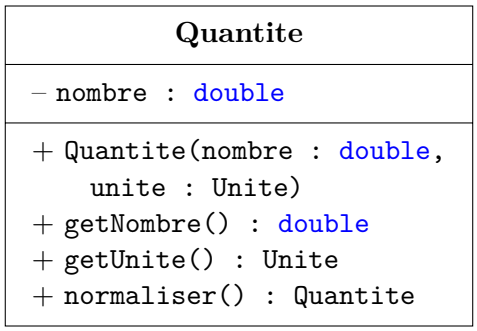
— Une recette est composée d'un nom et d'une description expliquant chaque étape de la recette.

2 Le `main.cpp`

Un fichier `main.cpp` vous est fournit. Il contient le `main` et différentes méthodes de test. Ce fichier doit être complété au fur et à mesure de l'avancée du TP afin de tester les différentes classes.

3 Méthodes utilitaires

Pour pouvoir manipuler les quantités, nous avons besoin de méthodes supplémentaires : la méthode `conversion` et une redéfinition de l'opérateur `+`. On met donc à jour le diagramme de la classe `Quantite`.



Codez ces deux nouvelles méthodes :

- La méthode `normaliser` permet de convertir des `KG` ou `G` en `KG`, des `ML`, `CL`, `DL` ou `L` en `L` et des `UNITE` en `UNITE`. Si la quantité courante est en `KG`, `L` ou `UNITE` alors une copie de cette quantité est retournée. Sinon, une nouvelle quantité équivalente est construite et retournée.
- L'opérateur `+` construit et retourne une nouvelle quantité représentant la somme des deux quantités additionnées. Si les deux quantités s'expriment dans la même unité, alors la quantité retournée aussi. Par exemple, la somme de `1dL` et `3dL` vaut `4dL`. Par contre, si les deux quantités ne s'expriment pas avec la même unité, on a deux cas :
  - Si les unités sont incompatibles (exemple : `G` avec `L` ou `KG` avec `UNITE`), la méthode affiche `"somme impossible : unités non compatibles"` et retourne la première quantité. Par exemple, la somme entre `10G` et `0.5L` retourne `10G`.
  - Si les unités sont compatibles (`G` et `KG` sont compatibles et `ML`, `CL`, `DL` et `L` sont compatibles) alors leur somme s'exprimera en `KG` ou `L`.

(Pour codez ces méthodes, vous pouvez, si vous le souhaitez, ajouter une méthode permettant de tester la compatibilité entre deux unités.)

4 Méthodes d'affichage

Nous voulons définir plusieurs méthodes permettant d'afficher des recettes. Ces méthodes n'appartiennent à aucune classe.

```
afficherNomEtIngredients(r : Recette, ingred : unordered_set<Ingredient>) : void
afficherTout(r : Recette, ingredients : unordered_set<Ingredient>) : void
afficherTout(recettes : map<Recette, unordered_set<Ingredient>>) : void
afficherNoms(recettes : map<Recette, unordered_set<Ingredient>>) : void
```

Coder ces 4 méthodes de façon à produire les affichages suivants :

— La méthode `afficherNomEtIngredients` permet d'afficher le nom et la liste d'ingrédients de la recette passée en paramètre comme dans l'exemple suivant :

```
Soupe de poireaux
        300g pommes de terre
        2 poireaux
        50cL eau
```

— La méthode `afficherTout(Recette,unordered_set<Ingredient>)` permet d'afficher le nom, la liste d'ingrédients puis la description de la recette passée en paramètre comme dans l'exemple suivant :

```
Soupe de poireaux
        300g pommes de terre
        2 poireaux
        50cL eau
description :
Faire cuire les légumes puis mixer.
```

— La méthode `afficherTout(map<Recette, unordered_set<Ingredient>>)` permet d'afficher toutes les recettes de la `map` passée en paramètre. Pour chacune de ces recettes, on affiche son nom, sa liste d'ingrédients puis sa description.

— La méthode `afficherNoms` permet d'afficher le nom de chaque recette de la `map` passée en paramètre. (Un peu comme un sommaire, mais sans numéro de page.)

Redéfinir l'opérateur `<<` dans certaines des classes concernées par ces méthodes d'affichage peut être utile. Pensez également à définir toutes les méthodes nécessaires au fonctionnement des conteneurs utilisés ici.

5 Namespace `appli` (suite)

Implémentez la classe `LivreRecettes`.
Celle-ci contient un attribut de type `map<Recette, unordered_set<Ingredient>>`.

— La méthode `ajouterRecette` permet d'ajouter la recette passée en paramètre au livre. Si cette recette est déjà présente (même nom, même description), alors la recette n'est pas ajoutée et le message `"recette déjà présente"` est affiché.

— `rechercherRecettes` recherche toutes les recettes (et leurs ingrédients) ayant pour nom celui donné en paramètre.

— `afficherNomsRecettes` affiche le nom de toutes les recettes du livre (un peu comme un sommaire)

— `afficher` affiche tout le contenu du livre (les recettes et leurs ingrédients).

— `rechercherRecettesAvec` recherche toutes les recettes (et leurs ingrédients) dont l'aliment passé en paramètre est un des ingrédients nécessaire à la recette.

— `faireListeCourses` permet de concevoir la liste de courses permettant de réaliser toutes les recettes passées en paramètre. Si une recette passée en paramètre n'est pas présente dans le livre, le message `"recette non trouvée"` est affiché.