Projet n°2 : Unités et quantités

Le but de ce projet est d'implémenter une bibliothèque de mesures physiques qui permet de coder les unités du système international avec le système de type de C++.

Étape 1 : Unités

La première partie consiste à définir une classe templatée `Unit` représentant une unité. De prime abord, la classe est déjà définie dans le projet et il semble qu'il n'y ait pas beaucoup de choses à faire. En fait, vous vous apercevrez qu'il sera nécessaire d'implémenter des types supplémentaires pour gérer les résultats des opérations. Il est conseillé de placer tous ces types dans un sous-espace de nom `details` puisqu'ils ne font pas partie de l'interface de la bibliothèque.

Il est fortement recommandé de lire la documentation de `std::ratio` dont vous aurez besoin dans la suite et qui peut vous servir d'inspiration pour cette partie du projet.

Étape 2 : Quantités

La deuxième partie consiste à implémenter une classe `Qty` représentant des quantités. Les quantités sont représentées par un entier, associé à un type unité et un ratio. Par exemple une quantité représentant des millimètres aura comme type `Qty<Metre, std::milli>`. Une quantité pourra être initialisée par une valeur ou alors sera initialisée à 0.

La difficulté ici consiste à implémenter correctement les opérateurs de comparaisons et les opérateurs arithmétiques, en prenant en compte les unités mais aussi les ratios.

Grâce à cette représentation, il est possible d'exprimer les unités de distance du système impériale (Mile, Yard, Foot, Inch). Il n'est pas nécessaire d'ajouter des nouvelles unités mais on peut les exprimer comme des quantités de distance.

Étape 3 : Opérateurs littéraux

La dernière partie consiste à ajouter des opérateurs littéraux pour le déclaration des quantités. Le projet permettra le déclaration de toutes les unités du système international mais aussi la déclaration des degrés Celsius et Fahrenheit.

Exemple d'utilisation

```cpp
#include "Units.h"

int main() {
  using namespace phy::literals;

  auto velocity = 100000_metres / 3600_seconds; // 100 km/h

  phy::Qty<Metre, std::milli> mm(32);
```

```
  auto nm = phy::qtyCast<phy::Qty<Metre, std::nano>>(mm);

}
```