Nombres rationnels

Exercice 1 Le but de cet exercice est de simuler les nombres rationnels. Ils sont caractérisés par un numérateur et un dénominateur, tous deux étant des nombres entiers.

1. Ecrire une classe nommée `Rat` contenant les deux membres de données `num_` et `den_` dans le fichier d'en-tête **rat.h**. Le compléter avec les déclarations des méthodes et fonctions ci-dessous.
2. Définir dans le fichier **rat.cpp** les méthodes et fonctions suivantes :
   (a) Le constructeur. Il faudra faire en sorte que si le numérateur est nul, l'objet courant soit le rationnel $\frac{0}{1}$, le dénominateur doit être toujours positif, et il faudra gérer le cas où le nombre rationnel n'est pas sous forme irréductible. De plus, `Rat r(8);` doit représenter le rationnel $\frac{8}{1}$.
   (b) Les accesseurs `get_num()` et `get_den()` pour récupérer la valeur des membres de données.
   (c) La surcharge des opérateurs internes suivants :
      i. `operator+=` pour un rationnel **ou** un entier (donc 2 surcharges).
      ii. `operator-=` pour un rationnel **ou** un entier.
      iii. `operator*=` pour un rationnel **ou** un entier.
      iv. `operator/=` pour un rationnel **ou** un entier.
   (d) La surcharge des opérateurs externes suivants :
      i. `operator<<` pour l'affichage (le rationnel $\frac{n}{1}$ aura pour affichage n seulement).
      ii. `operator+` pour un rationnel **ou** un entier comme opérandes (donc 3 surcharges).
      iii. `operator-` pour un rationnel **ou** un entier comme opérandes.
      iv. `operator*` pour un rationnel **ou** un entier comme opérandes.
      v. `operator/` pour un rationnel **ou** un entier comme opérandes.
      vi. `operator==` pour un rationnel **ou** un entier comme opérandes.
      vii. `operator!=` pour un rationnel **ou** un entier comme opérandes.
      viii. `operator<` pour un rationnel **ou** un entier comme opérandes.
      ix. `operator<=` pour un rationnel **ou** un entier comme opérandes.
      x. `operator>` pour un rationnel **ou** un entier comme opérandes.
      xi. `operator>=` pour un rationnel **ou** un entier comme opérandes.
3. Ecrire dans le fichier **rat_main.cpp** l'utilisation des méthodes et fonctions ci-dessus.

Dans tous les cas, utiliser l'espace de nom `ensiie` ainsi que les exceptions et les méthodes `const` quand elles sont nécessaires. On pensera à réutiliser le plus possible les méthodes ou fonctions déjà écrites.

Vecteurs

Exercice 2 Le but de cet exercice est de simuler des vecteurs de taille quelconque. On utilisera des pointeurs pour les données. Ces vecteurs sont donc caractérisés par un pointeur et une taille (le nombre d'éléments).

1. Ecrire une classe nommée `Vect` contenant les deux membres de données `data_` et `size_` dans le fichier d'en-tête **vect.h**. Le compléter avec les méthodes et fonctions suivantes.
2. Définir dans le fichier **vect.cpp** les méthodes et fonctions suivantes :
   (a) Le constructeur, qui prendra la taille du vecteur en argument.
   (b) Le destructeur.
   (c) le constructeur de copie.
   (d) L'accesseur `get_size()`.
   (e) La surcharge des opérateurs internes suivants :
      i. les 2 opérateurs `operator[]`.
      ii. `operator=`
   (f) La surcharge des opérateurs externes suivants :
      i. `operator<<` pour l'affichage sous la forme "(v1,...,vn)".
      ii. `operator+`
      iii. `operator-`
      iv. `operator*` pour le produit scalaire.
      v. `operator*` pour le produit d'un vecteur et un scalaire (et inversement).
      vi. `operator/` pour la division d'un vecteur par un scalaire.
   (g) la méthode `norm(double p)` pour la norme (include le fichier **cmath** pour utiliser les fonctions `fabs()` et `pow()`). On passera un argument pour la norme $p \geqslant 1$, aucun pour la norme 2.
3. Ecrire dans le fichier **vect_main.cpp** l'utilisation des méthodes et fonctions ci-dessus.

Dans tous les cas, utiliser l'espace de nom `ensiie` ainsi que les exceptions et les méthodes `const` quand elles sont nécessaires.