université PARIS-SACLAY

Travaux Pratiques de programmation

Cours de programmation modulaire

—Licence MI/IM/MNSI - Info 3—

Révisions: classes, tests, compilation séparée

Ce TP va vous servir à réviser les notions vues précédemment. L'objectif est de faire les exercices en appliquant la méthode de compilation séparée avec les fichiers `.hpp`, `.cpp` et `Makefile` ainsi que l'infrastructure de tests `doctest`. Si vous créez de nouveaux fichiers, **pensez dès la création à faire la commande**

```
git add nomDuFichier
```

Vous devez d'abord avoir fini le TP9 avant de passer à ce TP10. Ce TP10 sert de révisions pour ceux qui ont fini les TP précédents.

▸ **Exercice 1. (Gestion de réservation de courts de tennis)** On souhaite écrire une application pour gérer des réservations de courts de tennis. L'application permettra à un club de gérer un nombre quelconque de courts et il pourra préciser ses horaires d'ouverture (identiques pour tous ses courts). Les réservations se font par tranches d'une ou plusieurs heures. On s'occupe uniquement des réservations pour la journée courante, sans se soucier si une réservation est effectuée pour un créneau horaire dépassé. Les règles sont les suivantes :

- On ne peut pas réserver un créneau en dehors des heures d'ouverture ;
- Un adhérent peut réserver plusieurs créneaux dans la même journée mais il ne peut pas occuper plusieurs courts à une même heure ;
- Une réservation concerne toujours un unique court (on ne répartit pas sur plusieurs courts une demande de réservation qui ne tiendrait pas sur un seul court).

Les adhérents sont représentés par leur numéro (un entier strictement positif) : chaque adhérent a un numéro différent, supposé correct. On définit les types ci-dessous pour modéliser le planning d'occupation des courts. Chaque court du club sera désigné par un nom distinct, fixé à sa création.

```
1  struct Court {
2    string nom;
3    // Planning d'occupation de ce court:
4    // Chaque case correspond à une heure,
5    // en particulier la 1ere case correspond à l'heure d'ouverture du club.
6    // Pour une heure donnée, la case contient soit le numéro de l'adhérent
7    // qui a réservé ce court à cette heure, soit 0 si le court est libre
8    vector<int> occupation;
9  };
10
11 struct Club {
12   vector<Court> listeCourts;     // liste des courts du club
13   int hdebut;            // heure d'ouverture du club (ex. 8h)
14   int hfin;              // heure de fermeture du club (ex. 20h)
15 };
```

Pour chacune des fonctions ci-dessous, on n'oubliera pas d'écrire des tests et de les lancer ! ! !

1. Écrire un fichier `Makefile` permettant de compiler les fichiers `Club.cpp`, `Club.hpp` et `Club-main.cpp` fournis (ne pas modifier ces fichiers dans cette question). Compilez puis exécutez pour vérifier que cela fonctionne (indique 0 tests effectués, avec succès).
2. Écrire un constructeur pour un Club. On passe les horaires d'ouverture et de fermeture. Après exécution du constructeur, les horaires du club sont fixés mais le club ne contient encore aucun court. On supposera les heures correctes et l'heure de fermeture postérieure à l'heure d'ouverture.
3. Surcharger l'opérateur d'affichage pour le planning d'occupation d'un club [(1)] : pour chaque court on affiche son nom et pour chaque créneau occupé le numéro de l'adhérent concerné, en regroupant les créneaux successifs occupés par le même adhérent (exemple : 8h-11h : 25, 13h-14h : 32 pour un court qui ne serait occupé que de 8h à 11h par l'adhérent 25 et de 13h à 14h par l'adhérent 32).
4. Écrire une méthode d'ajout d'un court à un club. On passe en paramètre le nom du court. La méthode renvoie `true` si l'ajout a été possible, `false` sinon. Initialisez le planning d'occupation du court de façon à faciliter les réservations ultérieures.
5. Écrire une méthode de réservation d'un créneau sur un court donné. La méthode prend en paramètre un numéro d'adhérent, le nom du court, l'heure de démarrage et la durée du créneau. La méthode renvoie `true` si la réservation est possible et met à jour le planning d'occupation, et `false` sinon. Dans cette méthode, pour pouvoir réserver un court, on demande uniquement de vérifier qu'il ne soit pas déjà occupé (peu importe par qui et peu importe les autres courts).

   ```
   bool reserveCourt(string nom, int adhérent, int deb, int duree);
   ```

6. Écrire une méthode qui renvoie `true` si un adhérent donné a une réservation enregistrée à une heure donnée sur l'un des courts (parmi tous ceux du club). On suppose les paramètres corrects.
7. Écrire une méthode de réservation d'un créneau sur un court quelconque. La méthode prend en paramètre un numéro d'adhérent, l'heure de démarrage et la durée du créneau demandé. La méthode renvoie `true` si la réservation est possible sur un des courts et met à jour le planning d'occupation, et `false` sinon. La méthode doit procéder à l'ensemble des vérifications nécessaires.

   ```
   bool reserve(int adherent, int debut, int duree);
   ```

8. Écrire un programme principal qui appelle les différentes méthodes pour permettre à un utilisateur de réserver des cours ou de vérifier si un cours l'est déjà.

(1). On écrit cette méthode d'abord pour aider à comprendre comment accéder aux diverses informations. On suppose bien sûr que la méthode est appliquée à un club pour lequel les informations pertinentes auront été entrées grâce aux méthodes qui suivent.

▸ Exercice 2. (Encore plus de nombres !)

Voici quelques suggestions de travaux intéressants pour ceux qui vont vite :

Au TP7 nous avons travaillé sur les entiers naturels. Vous pouvez reprendre ce travail et l'utiliser dans une nouvelle classe `Relatif` qui code un entier relatif (positif ou négatif). Cette classe a deux attributs, la valeur absolue qui est de type `Naturel` et le signe qui sera d'un nouveau type énuméré.

Attention : il n'y a qu'un seul zéro et il est positif !

1. Créer deux fichiers `.hpp` et `.cpp` pour les entiers naturels à partir de ce que vous avez fait la semaine dernière.
2. Créer deux nouveaux fichiers `.hpp` et `.cpp` pour les entiers relatifs. On définira les mêmes méthodes et opérateurs que pour les naturels. Bien tout tester rigoureusement.
3. Plus difficile : surcharger l'opérateur de multiplication pour les naturels puis pour les relatifs. Bien tester.
4. Encore plus difficile : surcharger les opérateurs de division et de reste pour les naturels puis pour les relatifs. Bien tester.
5. En utilisant l'algorithme d'euclide (voir le fichier du cours sur les rationels), vous pouvez maintenant calculer les pgcd et les ppcm !
6. Si vous êtes arrivé ici, vous pouvez maintenant reprendre le fichier sur les rationnels vu en cours et l'adapter pour qu'il marche avec vos nombres naturels et relatifs : le numérateur est un relatif et le dénominateur un naturel non nul.