université PARIS-SACLAY

Travaux Pratiques de programmation n°4

Cours de programmation modulaire

—Licence MI/IM - Info 3—

Les tests avec `doctest`

Dans cette séance de travaux pratique, nous nous familiarisons avec l'infrastructure de test `doctest`. Cette infrastructure sera utilisée systématiquement dans les séances suivantes. Il est donc important de ne pas se contenter de répondre aux questions mais aussi d'explorer ce concept et son utilisation pour voir ce qui est faisable et ce qui ne l'est pas.

Nous en profiterons pour revoir les `structs` et la surcharge d'opérateurs.

▸ **Exercice 1. (On fait des tests avec `doctest`)**

1. Ouvrir le fichier `puissance.cpp` et coder la fonction `int puissance(int nombre, int exposant)` qui calcule la puissance de `nombre` par la valeur de `exposant`.
2. Pour pouvoir faire des tests avec `doctest`, il suffit que le fichier `doctest.h` soit dans le même répertoire que le fichier `puissance.cpp`.
   — Remarquer les deux lignes du fichier `puissance.cpp` qui permettent d'inclure `doctest` :

```
1 #define DOCTEST_CONFIG_IMPLEMENT
2 #include "doctest.h"
```

   — Remarquer aussi les 4 lignes à partir du `main` qui permettent de lancer tous les tests `doctest` :

```
1 int main(int argc, const char** argv){
2   doctest::Context context(argc, argv);
3   int test_result = context.run();
4   if (context.shouldExit()) return test_result;
```

   — Enfin, remarquer les deux lignes qui permettent de définir l'opération de test ainsi qu'une proposition de test :

```
1 TEST_CASE("Test de la fonction puissance") {
2    CHECK(puissance(10, 0) == 1);
```

3. Proposer d'autres tests pertinents avec `doctest` pour la fonction `int puissance(int nombre, int exposant)`.
4. Compiler et exécuter votre programme. Regarder et analyser ce que votre programme affiche.
5. Ajouter un test volontairement faux, compiler et lancer les tests. Vérifier que l'erreur est bien reportée.
6. Remplacer le `CHECK` du test faux par un `CHECK_FALSE`, recompiler et vérifier que l'erreur n'apparaît plus. Note : ce test avec `CHECK_FALSE` est très probablement inutile ici. On va néanmoins le conserver pour l'exemple.
7. Exécuter votre programme avec la commande `./puissance -h`. Cela va afficher l'aide (help) de toutes les options que vous pouvez essayer.
8. Essayer en particulier ce que font les appels `./puissance -s` et `./puissance -d`.

▸ **Exercice 2. (Surcharge et tests `doctest` de fonctions pour les dates)** Vous avez déjà codé plusieurs fonctions sur le sujet des dates dans la première séance de TP. On va donc se baser sur ce que vous aviez codé pour effectuer des surcharges d'opérateurs pour la manipulation des dates et utiliser ensuite `doctest` pour réaliser des tests sur un certain nombre de fonctions.

On vous demande d'ouvrir le fichier `date-doctest.cpp` et de :

1. Coder la surcharge de l'opérateur d'affichage `<<` pour afficher une `Date` sous le format jj/mm/aaaa.
2. Le `main` fourni contient des affichages de dates. Compiler et exécuter votre programme pour vérifier votre opérateur d'affichage. Comme `doctest` lance tous les tests y compris ceux des fonctions que vous n'avez pas encore écrites, `doctest` signale qu'un certains nombre de tests ont échoués, mais vous devez voir ensuite s'afficher deux dates.

Les fonctions suivantes seront testées directement avec `doctest` au fur et à mesure en vérifiant que les tests correspondant passent.

3. Coder la surcharge de l'opérateur `==` pour vérifier que deux dates `d1` et `d2` sont égales.
4. Coder la surcharge de l'opérateur `!=` pour vérifier que deux dates `d1` et `d2` sont différentes.
5. On vous a donné un exemple de test `doctest` pour ces deux dernières surcharges, proposer d'autres tests pertinents.
6. Proposer des tests pertinents pour la fonction fournie `bool estBissextile(int annee)`.
7. Proposer des tests pertinents pour la fonction fournie `int nbJourMois(int mois, int annee)`. Parmi les tests, il faut aussi vérifier que si le mois est invalide (-1, 0 où 13 par exemple), une exception est bien levée. Pour ceci, on utilise `CHECK_THROWS_AS`.
8. Coder la surcharge de l'opérateur de lecture `>>` pour lire une Date (jj mm aaaa). Il faut bien évidemment vérifier que la date lue est correcte en utilisant la fonction fournie `bool estCorrecteDate(d)` qui vérifie si la date `d` est correcte. Si ce n'est pas le cas, on lèvera une exception.
9. Dé-commenter dans le main les lignes qui font la saisie de la date `aujourdhui` et vérifier que tout marche bien.
10. Coder la surcharge de l'opérateur `<` qui dit si la date `d1` est avant la date `d2`. On demande de n'utiliser ni boucle ni la fonction lendemain.
11. Proposer des tests pertinents pour la fonction de surcharge précedente `<`.
12. Coder la surcharge de l'opérateur `+` qui ajoute un nombre de jours `n` à une date `d`. On supposera que `n` est positif et lèvera une exception si ce n'est pas le cas. On pourrait faire une version simple en utilisant la fonction lendemain, mais elle ne serait pas très efficace. Il est mieux de faire une version sans appel à lendemain (et sans passer par tous les jours de `d` à `d+n`).
13. Proposer des tests pertinents pour l'opérateur `+`.
14. Coder la surcharge de l'opérateur `-` qui calcule le nombre de jours écoulés entre les dates `d2` et `d1`. On supposera que `d1` est après `d2` et levera une exception si ce n'est pas le cas.
15. Proposer des tests pertinents pour l'opérateur `-`.
16. Proposer des tests pertinents pour la fonction `int jourDate(Date d)` qui retourne le jour de la semaine d'une date (renvoie 0 pour lundi, 1 pour mardi, ...).

▸ **Exercice 3.** S'il vous reste du temps, reprendre le jeu d'échec du TD 3, le compléter et ajouter des tests avec `doctest` (voir exercices 2 et 5 du TD3, à compléter aussi selon vos propres idées).

Pour installer `doctest` dans le répertoire du TD 3 il suffit de copier le fichier `doctest.h` et de l'ajouter au fichiers pris en compte par Git et les scripts Travo avec

```
git add doctest.h
```

De même, vous pouvez reprendre la calcul de racine carrée du TD 3, le compléter et ajouter des tests avec `doctest` (voir exercices 3 et 4 du TD3).