
TD 3

Une classe Date

3.1 Objectif

Il s'agit là de la première classe C++ que vous allez définir complètement. Sa complexité est relativement faible, assez similaire à celle de la classe `Rational` vue en cours. Comme cette dernière, c'est une classe qui définit des opérateurs arithmétiques et dont le constructeur réalise une sorte de normalisation (au moins une vérification).

Pour vous éviter de perdre du temps à développer des algorithmes dont l'intérêt est en l'occurrence limité, une partie du code vous est fourni (dans le répertoire `Date`), mais il est écrit en C. Cet exercice constitue donc aussi une occasion de comparer la réalisation du même type abstrait en C et en C++.

3.2 Description de la classe `Date`

Votre mission est de définir une classe `Date` représentant des dates du calendrier occidental et permettant de les manipuler (opérations arithmétiques, entrées-sorties). On se limitera aux années postérieures à 1900¹ (incluse).

La classe `Date` a un constructeur par défaut : la date créée est celle du jour courant (aujourd'hui) ; elle possède aussi un constructeur prenant trois entiers (année, mois, jour) :

```
Date dnow, d;           // la date courante (aujourd'hui)
Date d14J(2002, 7, 14); // le 14 juillet 2002
```

Bien entendu, ce dernier constructeur doit vérifier que les trois entiers proposés forment bien une date valide et, dans le cas contraire, lever des exceptions.

Vous définirez aussi des fonctions d'accès permettant de consulter l'année, le mois et le jour d'une date.

Vous devez également définir des opérateurs permettant d'ajouter ou de soustraire un nombre entier de jours à une date² :

```
d = dnow + 1;    // demain
d = dnow - 1;    // hier
d = dnow + 365;  // dans un an, aujourd'hui
```

On doit aussi pouvoir soustraire deux dates, ce qui retourne le nombre *algébrique* de jours entre les deux dates. Par définition *demain - aujourd'hui = 1* et *aujourd'hui - demain = -1*.

```
int n = d14J - dnow; // jours entre le 14 juil 2002 et aujourd'hui
n = (dnow + 1) - dnow; // 1, par définition
n = dnow - (dnow + 1); // -1
n = Date(2003, 2, 10) - Date(2003, 6, 12); // -122
```

1. Ceci à cause d'une limitation de la norme POSIX qui ne considère pas de date antérieure à 1900.

2. Avec un peu d'aide, il vous sera même possible de définir des opérateurs comme `+=`, `-=`, `++` et `--`.

Remarquer, dans le dernier exemple, comment l'appel explicite du constructeur permet de créer « à la volée » un objet *temporaire* de type `Date`.

Les dates doivent pouvoir être comparées avec les opérateurs relationnels habituels (`==`, `!=`, `<`, `>`, `<=`, `>=`).

Enfin les dates doivent supporter les habituels opérateurs d'entrée-sortie des `iostreams`. L'opérateur d'affichage (`<<`) doit produire une forme lisible de la date. Ainsi:

```
cout << d14J << endl;
```

doit afficher sur la sortie standard

```
July 14, 2002
```

(ou son équivalent français). L'opérateur de lecture (`>>`), quant à lui, reçoit des dates composées d'entiers sous la forme `yyyy/mm/dd`. Il doit bien entendu refuser de lire des dates invalides.

3.3 Note sur le code C fourni

Le répertoire `Date` contient le code C réalisant un type abstrait date assez rustique. On trouvera trois fichiers de source C : `date_c.h`, la définition de la structure `date`, `date_c.c`, son implémentation, et `main_date_c.c`, un programme de test. Une `Makefile` permet de reconstruire l'exécutable de test (faire `make tst_date_c.exe`). Cette `Makefile` est également prête à accueillir le code que vous développerez, à condition que le fichier de définition de votre classe `Date` se nomme `Date.h`, son implémentation `Date.cpp`, et le programme de test `main_Date.cpp`. Il suffira alors de faire `make` et l'exécutable produit se nommera `tst_Date.exe`. Si vous ne respectez pas ces noms, il vous faudra éditer la `Makefile`.

Le code de la `struct date` vise juste à fournir les algorithmes pour calculer la date du jour et ajouter ou soustraire un nombre de jours à une date. Vous devrez inventer celui pour soustraire deux dates. Essayez de le faire économiquement (en code) et avec la même approche que celle utilisée dans le code fourni.

Ce code C n'est certes pas le code définitif d'un vrai type abstrait date. En particulier aucune vérification n'est effectuée. Ceci dit, les vérifications nécessaires sont plus difficiles à imposer en C qu'en C++. Ceci est aussi une leçon à tirer de l'exercice.

Noter enfin que le code C (et sans doute votre code C++) nécessite a priori un système compatible avec la norme POSIX (Unix, Linux, Cygwin...) pour trouver la date du jour (fonction `POSIX localtime()` utilisée dans `today()`). Cependant, la plupart des environnements C, même non POSIX, en particulier Visual C++ 7 (et sans doute 6), supportent la fonction `localtime()`.