CPPLI : TD 8 : C++ : Classe

Nicolas Vansteenkiste Romain Absil Jonas Beleho *
(Esi – He2b)

Année académique 2019 – 2020

Ce TD [1] aborde la création et l'utilisation de classes [2] en C++.
Réalisez ce TD en exploitant le standard C++17 [3].

1 Énumération fortement typée `Sign`

Le fichier `sign_incomplete.h` [4] contient la définition de l'`enum class Sign` ainsi que celles de diverses fonctions qui l'utilisent, mais *sans* les implémentations de ces dernières.

Toute la documentation de l'énumération fortement typée [5] (*strongly typed enumeration*) `Sign` est disponible au format html, après décompression du fichier `html.7z` [6].

Ex. 8.1 Lisez la documentation de l'`enum class Sign` et de ses fonctions.

---

*Et aussi, lors des années passées : Monica Bastreghi, Stéphan Monbaliu, Anne Rousseau et Moussa Wahid.

1. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/td08_cpp.pdf (consulté le 27 novembre 2019).
2. https://en.cppreference.com/w/cpp/language/class (consulté le 24 novembre 2019).
3. https://isocpp.org/search/google?q=c%2B%2B17 (consulté le 27 novembre 2019).
4. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/sign_incomplete.h (consulté le 27 novembre 2019).
5. https://docs.microsoft.com/fr-fr/cpp/cpp/enumerations-cpp?view=vs-2019 (consulté le 24 novembre 2019).
6. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/html.7z (consulté le 27 novembre 2019).

Ex. 8.2 Prenez le fichier `sign_incomplete.h` et :

1. renommez-le en `sign.h` ;
2. adaptez là où nécessaire son contenu à son nouveau nom ;
3. déplacez l'énumération `Sign` et ses fonctions dans l'espace de noms[7] (ou de nommage) `gxxxxx` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez les fonctions marquées d'un commentaire `// TODO` de sorte qu'elles respectent les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

2 Classe `Fraction`

Le fichier `fraction_incomplete.h`[8] contient la définition de la `class Fraction` ainsi que celles de diverses fonctions qui l'utilisent. Les implémentations de celles-ci ainsi que des méthodes de `Fraction` manquent dans les fichiers `fraction_incomplete.h` et `fraction_incomplete.cpp`[9].

La documentation de cette classe et de ces fonctions est disponible au format html après décompression du fichier `html.7z`.

Ex. 8.3 Lisez la documentation de la `class Fraction` et de ses fonctions.

Ex. 8.4 Prenez les fichiers `fraction_incomplete.h` et `fraction_incomplete.cpp` et :

1. renommez-le en `fraction.h` et `fraction.cpp`, respectivement ;
2. adaptez leurs contenus à leurs nouveaux noms, ainsi qu'au nouveau nom donné au fichier `sign_incomplete.h` à l'exercice 2 ;
3. déplacez la classe `Fraction` ainsi que ses fonctions dans l'espace de noms `gxxxxx` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez les fichiers `fraction.h` et `fraction.cpp` aux endroits marqués d'un commentaire `// TODO` de sorte que leurs contenus respectent les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

7. https://en.cppreference.com/w/cpp/language/namespace (consulté le 24 novembre 2019).
8. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/fraction_incomplete.h (consulté le 27 novembre 2019).
9. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/fraction_incomplete.cpp (consulté le 27 novembre 2019).

2.1 Variante `constexpr`

Le fichier `fraction_constexpr_incomplete.h`[10] contient la définition d'une variation de la `class Fraction`. Dans cette variante, toutes les méthodes et toutes les fonctions sont marquées `constexpr`[11]. Les implémentations des méthodes et fonctions sont manquantes dans le fichier `fraction_constexpr_incomplete.h`.

La documentation de cette classe et de ces fonctions est disponible au format html après décompression du fichier `html_constexpr.7z`[12].

Ex. 8.5 Prenez le fichier `fraction_constexpr_incomplete.h` et :

1. renommez-le en `fraction_constexpr.h` ;
2. adaptez son contenu à son nouveau nom, ainsi qu'au nouveau nom donné au fichier `sign_incomplete.h` à l'exercice 2 ;
3. déplacez la classe `Fraction` ainsi que ses fonctions dans l'espace de noms `gxxxxx` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez le fichier `fraction_constexpr.h` aux endroits marqués d'un commentaire `// TODO` de sorte que son contenu respecte les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

3 Mise en œuvre

Les fichiers `data_fraction.h`[13] et `data_fraction.cpp`[14] contiennent les définitions et implémentations de deux fonctions, `data_signed()` et `data_unsigned()`, pour la génération de fractions. Les prototypes de ces fonctions sont précédés de quelques commentaires qu'il vous est conseillé de lire.

3.1 Variante `signed`

Ex. 8.6 Utilisez la fonction `data_signed()` pour tester le constructeur de `Fraction` à deux arguments `int`. Pour ce faire, invoquez cette fonction et utilisez toutes les `std::pair<int, int>` qu'elle retourne pour garnir un `std::vector`[15] de `Fraction`. Cependant, tenez à jour un compteur des `std::pair`[16] générant une erreur lors de

10. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/fraction_constexpr_incomplete.h (consulté le 27 novembre 2019).
11. https://en.cppreference.com/w/cpp/language/constexpr (consulté le 24 novembre 2019).
12. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/html_constexpr.7z (consulté le 27 novembre 2019).
13. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/data_fraction.h (consulté le 27 novembre 2019).
14. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td08_cpp/ressource/data_fraction.cpp (consulté le 27 novembre 2019).
15. https://en.cppreference.com/w/cpp/container/vector (consulté le 27 novembre 2019).
16. https://en.cppreference.com/w/cpp/utility/pair (consulté le 27 novembre 2019).

l'instanciation d'une fraction.

Affichez le compteur d'erreurs, la taille du `std::vector<Fraction>` et le contenu de ce dernier.

Voici un affichage possible correspondant à cet exercice :

```
error_count: 7
fractions.size(): 43

fractions content:
7/6 -4 0 -7/3 3/7 -5/7 -3/7 1/3 -1/3 1/6 -3 -1/3 3/2 4/7 -4/3 1/2
-7/2 -4/7 3/4 -1/3 1/3 3 4/3 2/5 0 -6/7 0 2/3 -7/6 0 -1/2 0 -1 1
-5/6 -7/3 3 4/3 7/6 4/3 1/5 -1/5 3
```

3.2 Tri via pointeurs

Dans l'exercice qui suit, il s'agit de trier dans l'ordre croissant les fractions du `std::vector<Fraction>` obtenu à l'Ex. 8.6. Si on tente de le trier directement avec l'algorithme `std::sort()` [17], un gros problème se pose ! La classe `Fraction` est immuable car tous ses attributs sont `const`. Il est dès lors impossible de réassigner les contenus des cellules du `std::vector` de `Fraction` et donc de les modifier ou encore de trier le `std::vector`.

Il existe heureusement des parades. On peut construire, élément par élément, un nouveau `std::vector` dont le contenu est identique à celui à trier si ce n'est qu'il est précisément trié. C'est fastidieux.

Une parade alternative consiste à produire un `std::vector` de *pointeurs* de `Fraction`. Le premier élément du `std::vector` de pointeurs pointe sur la première fraction du `std::vector` de `Fraction`, le deuxième pointeur sur la deuxième `Fraction`, etc. jusqu'au dernier pointeur pointant sur la dernière `Fraction` du `std::vector` de `Fraction`. Ensuite, à la place de trier le `std::vector` de `Fraction` — opération impossible —, on trie le `std::vector` de pointeurs. En parcourant alors le `std::vector` de pointeurs trié et en déréférençant ceux-ci, le résultat est similaire au tri du `std::vector` de `Fraction` ! C'est cette approche que nous adoptons dans l'exercice suivant.

Ex. 8.7 À la suite de votre code réponse à l'Ex. 8.6, produisez un `std::vector` de pointeurs de `const Fraction` dont chaque élément pointe sur l'élément de même index du `std::vector` de `Fraction` construit initialement. Les pointeurs sont de type `const Fraction *` pour rendre impossible toute modification par inadvertance des `Fraction` du premier `std::vector`. L'algorithme `std::transform()` [18] peut être utile.

Affichez le déréférencement du contenu du `std::vector` de pointeurs avant le tri, triez les pointeurs dans l'ordre croissant des fractions pointées puis affichez le déréférencement du contenu du `std::vector` de pointeurs après le tri.

17. https://en.cppreference.com/w/cpp/algorithm/sort (consulté le 24 novembre 2019).
18. https://en.cppreference.com/w/cpp/algorithm/transform (consulté le 24 novembre 2019).

Avec les mêmes données que celles de l'Ex. 8.6, voici un affichage possible :

```
before sorting (pointer):
7/6 -4 0 -7/3 3/7 -5/7 -3/7 1/3 -1/3 1/6 -3 -1/3 3/2 4/7 -4/3 1/2
-7/2 -4/7 3/4 -1/3 1/3 3 4/3 2/5 0 -6/7 0 2/3 -7/6 0 -1/2 0 -1 1
-5/6 -7/3 3 4/3 7/6 4/3 1/5 -1/5 3

after sorting (pointer):
-4 -7/2 -3 -7/3 -7/3 -4/3 -7/6 -1 -6/7 -5/6 -5/7 -4/7 -1/2 -3/7
-1/3 -1/3 -1/3 -1/5 0 0 0 0 0 1/6 1/5 1/3 1/3 2/5 3/7 1/2 4/7 2/3
3/4 1 7/6 7/6 4/3 4/3 4/3 3/2 3 3 3
```

3.3 Variante `unsigned`

Ex. 8.8 Utilisez la fonction `data_unsigned()` des fichiers `data_fraction.h` et `data_fraction.cpp` pour tester le constructeur de `Fraction` à trois arguments : un `Sign` et deux `unsigned`. Pour ce faire, invoquez cette fonction et utilisez tous les `std::tuple<int, unsigned, unsigned>` qu'elle retourne pour garnir un `std::vector` de `Fraction`[19]. Cependant, tenez à jour un compteur des `std::tuple`[20] générant une erreur lors de l'instanciation d'une fraction.

Affichez le compteur d'erreurs, la taille du `std::vector<Fraction>` et le contenu de ce dernier.

Voici un affichage possible correspondant à cet exercice :

```
error_count: 33
fractions.size(): 67

before sorting:
-1/6 0 1/2 1 0 -7/10 -1/3 7 1 3/5 -1 1/2 -6 -5/4 0 5/3 2/9 8/5
3/2 1 10/9 -1 -2/3 1/2 7/8 1 -7/5 7/2 1 1/2 -5/9 -1 0 3/4 7/2
7/9 0 -2/5 2 3/8 4/3 0 2/9 -2/3 -9 1 -1/3 8/7 7/6 -3 -7 5/6 -1/5
4/3 3 -2 7/5 7/10 -7/3 -3/7 2/9 5/8 -1/7 2/3 10/7 -8 -9/8
```

3.4 Tri via `reference_wrapper`

Dans l'exercice qui suit, il s'agit de trier dans l'ordre décroissant les fractions du `std::vector<Fraction>` obtenu à l'Ex. 8.8.

19. Vous pouvez éventuellement purger et recycler le `std::vector<Fraction>` de l'Ex. 8.6.
20. https://en.cppreference.com/w/cpp/utility/tuple (consulté le 27 novembre 2019).

Plutôt que de trier ce `std::vector` à travers un `std::vector<const Fraction *>` comme dans la section 3.2, nous allons ici mettre en œuvre une nouvelle parade. Cette alternative consiste à produire un `std::vector` de `std::reference_wrapper`[21] de `Fraction`. Le premier élément du `std::vector` de `std::reference_wrapper` réfère la première fraction du `std::vector` de `Fraction`, le deuxième `std::reference_wrapper` la deuxième `Fraction`, etc. jusqu'au dernier `std::reference_wrapper` référant la dernière `Fraction` du `std::vector` de `Fraction`. Ensuite, à la place de trier le `std::vector` de `Fraction` — opération impossible —, on trie le `std::vector` de `std::reference_wrapper`. Ceci est possible car contrairement aux *lvalue references*[22], les `std::reference_wrapper` se détachent de l'objet référencé par leur opérateur d'assignation[23].

Ex. 8.9 À la suite de votre code réponse à l'Ex. 8.8, produisez un `std::vector` de `std::reference_wrapper<const Fraction>` dont chaque élément réfère l'élément de même index du `std::vector` de `Fraction` construit initialement. Ce sont des `const Fraction` qui sont enveloppées pour rendre impossible toute modification par inadvertance des `Fraction` du premier `std::vector`. L'algorithme `std::copy()`[24] peut être utile.

Affichez le déréférencement du contenu du `std::vector` de `std::reference_wrapper` avant le tri, triez les `std::reference_wrapper` dans l'ordre décroissant des fractions référencées puis affichez après le tri le déréférencement du contenu du `std::vector` de `std::reference_wrapper`.

Avec les mêmes données que celles de l'Ex. 8.8, voici un affichage possible :

```
 before sorting (reference_wrapper):
-1/6 0 1/2 1 0 -7/10 -1/3 7 1 3/5 -1 1/2 -6 -5/4 0 5/3 2/9 8/5
 3/2 1 10/9 -1 -2/3 1/2 7/8 1 -7/5 7/2 1 1/2 -5/9 -1 0 3/4 7/2
 7/9 0 -2/5 2 3/8 4/3 0 2/9 -2/3 -9 1 -1/3 8/7 7/6 -3 -7 5/6 -1/5
 4/3 3 -2 7/5 7/10 -7/3 -3/7 2/9 5/8 -1/7 2/3 10/7 -8 -9/8

 after sorting (reference_wrapper):
 7 7/2 7/2 3 2 5/3 8/5 3/2 10/7 7/5 4/3 4/3 7/6 8/7 10/9 1 1 1 1
 1 1 7/8 5/6 7/9 3/4 7/10 2/3 5/8 3/5 1/2 1/2 1/2 1/2 3/8 2/9 2/9
 2/9 0 0 0 0 0 0 -1/7 -1/6 -1/5 -1/3 -1/3 -2/5 -3/7 -5/9 -2/3 -2/3
-7/10 -1 -1 -1 -9/8 -5/4 -7/5 -2 -7/3 -3 -6 -7 -8 -9
```

21. https://en.cppreference.com/w/cpp/utility/functional/reference_wrapper (consulté le 24 novembre 2019).
22. https://en.cppreference.com/w/cpp/language/reference#Lvalue_references (consulté le 24 novembre 2019).
23. https://en.cppreference.com/w/cpp/utility/functional/reference_wrapper/operator%3D (consulté le 24 novembre 2019).
24. https://en.cppreference.com/w/cpp/algorithm/copy (consulté le 24 novembre 2019).