CPPLI : TD 5 : C++ : Classes : Rudiments

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

Année académique 2017 – 2018

Ce TD [1] aborde les rudiments de la création de classes en C++.
Durée : 2 séances.

1. `class Date`

Le répertoire `date_forstudent` [2] fourni avec cet énoncé contient :

— le fichier `date.doxyfile` [3] de configuration pour produire une documentation à l'aide de doxygen [4] ;

— le fichier `date.pro` [5] de configuration d'un projet pour Qt Creator (et qmake) ;

*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/td05_cpp/td05_cpp.pdf
2. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/
3. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/date.doxyfile
4. http://www.stack.nl/~dimitri/doxygen/
5. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/date.pro

— le fichier source `main.cpp`[6] faisant partie du projet `date.pro` et contenant l'implémentation incomplète de sa fonction principale, reproduit en annexe A.1 ;

— le fichier d'en-têtes `date.h`[7] et le fichier source `date.cpp`[8], reproduits en annexe A.2 et A.3, respectivement, faisant également partie de ce projet et contenant la définition et l'implémentation partielle de `Date` ;

— un répertoire `doc`[9] dont le contenu est :

  — une documentation de la classe `Date` au format pdf, `date_refman.pdf`[10] ;

  — un répertoire `html`[11] contenant une documentation de la classe `Date` au format html et dont le point d'entrée est la page index.html[12]

Le répertoire `validation`[13] quant à lui contient le fichier `validation.hpp`[14]. Celui-ci fait également partie du projet `date.pro` car les fonctions qui y sont implémentées sont utilisées par la classe `Date`. Il est reproduit en annexe B.

Ex. 5.1 Lisez la documentation de la classe `Date`, dans sa version pdf ou html.

Ex. 5.2 Ouvrez le projet `date.pro` dans Qt Creator. Si ce n'est déjà fait, activez le plugin Todo[15] comme expliqué en fin de TD 0[16]. Dans la vue *To-Do Entries*, sélectionnez *Active Project*. Treize (13) entrées apparaissent. Éliminez-les une à une de sorte à produire une classe `Date` et des fonctions de manipulation de `Date` en accord avec la documentation fournie.

Testez vos implémentations de méthodes et fonctions dans `main` une à une et au fur et à mesure que vous les réalisez. La première méthode dont il faut s'occuper est donc le constructeur de `Date`.

Pour réaliser cet exercice, il est interdit de modifier les fichiers `main.cpp`, `date.h` et `date.cpp` ailleurs que là où se trouvent les commentaires `// TODO`.

6. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/main.cpp
7. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/date.h
8. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/date.cpp
9. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/doc
10. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/doc/date_refman.pdf
11. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/doc/html
12. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/doc/html/index.html
13. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/validation/
14. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/validation/validation.hpp
15. https://katecpp.github.io/qtcreator-todo/
16. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td00_c_cpp/td00_c_cpp.pdf

Ex. 5.3 Ouvrez le fichier `date.doxyfile` dans doxywizard [17], le programme à interface graphique pour doxygen. Modifiez `date.doxyfile` de sorte que :

— le(s) fichier(s) d'en-têtes apparaissent dans la documentation ;

— seule une documentation html soit générée ;

— les membres privés n'apparaissent pas dans la documentation.

Produisez alors une documentation pour la classe `Date`.

Remarque Pour lier la documentation que vous produisez avec celle en ligne du site de référence `http://en.cppreference.com/w`, vous devez récupérer le dossier `cppreferencetagfile` [18]. Il contient le fichier `cppreference-doxygen-web.tag.xml` [19]. Celui-ci permet à doxygen de faire le lien, à condition que le fichier de configuration de votre documentation soit bien paramétré.

Voici comment procéder. Dans doxywizard, sélectionnez l'onglet *Expert*. Choisissez alors l'avant-dernier *Topics* nommé *External*. Cliquez su le bouton *Browse to a file* à droite de la zone d'édition elle-même à droite de l'étiquette *TAGFILES*. Naviguez jusqu'à l'emplacement où vous avez sauvegardé le fichier `cppreference-doxygen-web.tag.xml` et sélectionnez-le. Dans la zone d'édition ajoutez `=http://en.cppreference.com/w/` juste à droite de *`chemin relatif`*`/cppreference-doxygen-web.tag.xml` de sorte à obtenir l'expression :

`chemin relatif``/cppreference-doxygen-web.tag.xml=http://en.cppreference.com/w/`

dans la zone d'édition à droite de *TAGFILES*. Cliquez enfin sur le bouton + pour ajouter cette configuration.

Une capture d'écran [20] présente dans le répertoire `cppreferencetagfile` sous la forme d'un fichier png illustre ceci.

Ex. 5.4 Déplacez la classe `Date` dans un espace de nom `namespace gxxxxx` où `xxxxx` est votre numéro d'étudiant.

Documentez votre espace de nommage et produisez la documentation.

17. http://www.stack.nl/~dimitri/doxygen/manual/doxywizard_usage.html
18. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/cppreferencetagfile/
19. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/cppreferencetagfile/cppreference-doxygen-web.tag.xml
20. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/cppreferencetagfile/doxywizard_configuration_externaldoc_tag.png

2. `enum class Sex`

Ex. 5.5 Implémentez, dans *votre* espace de nom, l'énumération fortement typée [21] nommée `Sex` et les fonctions utilitaires telles que décrites dans le document `sex_refman.pdf` [22] ou en hypertexte [23].

Notez que la fonction de conversion d'une valeur de `Sex` vers une `std::string` utilise l'énumération fortement typée `FormOption` définie dans le fichier `formoption.h` [24] reproduit en annexe C.

Documentez l'énumération `Sex` et ses fonctions, puis produisez cette documentation.

3. `class Person`

Ex. 5.6 Il s'agit maintenant de produire la classe `Person`. Elle est munie d'attributs de types :

- `gxxxxx::Date` [25] (voir section 1) pour représenter sa date de naissance ;
- `gxxxxx::Sex` [26] (voir section 2) pour représenter son sexe ;
- `std::string` [27] pour représenter son nom de famille ;
- `std::vector` [28] de `std::string` pour représenter ses prénoms.

Implémentez, dans *votre* espace de nom, la classe `Person` et les fonctions utilitaires telles que décrites dans le document `person_refman.pdf` [29] ou en hypertexte [30].

Documentez cette classe et ses fonctions, puis produisez cette documentation.

Remarque Notez qu'ici tous les attributs sont connus lors de la création d'un objet `Person`. Qu'en est-il si on ajoute des attributs pour la date de décès ou les éventuelles dates de mariage et les liens vers les conjoints ? Nous aborderons ces problèmes plus tard.

21. http://cplusplus-development.blogspot.be/2013/04/c-11-enum-class-scoped-and-strongly.html
22. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/sex_forstudent/sex_refman.pdf
23. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/sex_forstudent/html/index.html
24. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/misc/formoption.h
25. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/date_forstudent/doc/html/classnvs_1_1_date.html
26. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/sex_forstudent/html/namespacenvs.html#aec1700e0e891785574c3445c9c9d66c5
27. http://en.cppreference.com/w/cpp/string/basic_string
28. http://en.cppreference.com/w/cpp/container/vector
29. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/person_forstudent/person_refman.pdf
30. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/person_forstudent/html/index.html

Nous nous concentrons ici sur le b.a. ba de l'écriture de classes en C++. Les notions intermédiaires liées par exemple aux attributs dynamiques ou à l'héritage sont étudiées dans un prochain TD.

4. Conteneur de `class Person`

4.1. Génération

Le fichier d'en-têtes `testgenerator.h`[31] contient le prototype :

```
std::vector<std::tuple<std::vector<std::string>, std::string, char,
  std::tuple<int, unsigned, unsigned>>> people(unsigned size = 1000);
```

Son code source est présent dans le fichier `testgenerator.cpp`[32]. Ces deux fichiers sont reproduits en annexe D.

En gros, elle fournit les arguments pour construire un ensemble de `Person`. Pour plus de détails, référez-vous à sa documentation dans son fichier d'en-tête.

Remarquez qu'elle utilise les fonctions de génération de nombres aléatoires définies dans le fichier `random.hpp`[33] reproduit en annexe E.

Ex. 5.7 Utilisez la fonction `people()` mentionnée ci-dessus pour tester les constructeurs de `Person` que vous avez produits. Pour ce faire, invoquez `people()` et, pour chaque `std::tuple`[34] d'arguments de construction d'une personne obtenu, ajoutez à l'aide d'`emplace_back()`[35] une `Person` à un `std::vector` construit vide. Ensuite, affichez le contenu du `std::vector` de `Person` et testez ainsi l'opérateur d'injection d'une personne dans un flux en sortie ou la fonction de conversion d'une personne en chaîne de caractères.

Voici un début d'affichage possible correspondant à cet exercice :

```
Nom               : Bah
Prénom(s)         : David Imran Mohamed Youssef
Date de naissance : 27/09/2014
Sexe              : m

Nom               : Bah
Prénom(s)         : Mohamed Youssef Gabriel Yanis Lucas Adam Amir
Date de naissance : 26/05/2015
```

31. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/testgenerator/testgenerator.h
32. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/testgenerator/testgenerator.cpp
33. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td05_cpp/random/random.hpp
34. http://en.cppreference.com/w/cpp/utility/tuple
35. http://en.cppreference.com/w/cpp/container/vector/emplace_back

```
Sexe                : m

Nom                 : Mertens
Prénom(s)           : Aya Lina Nour
Date de naissance : 26/05/2013
Sexe                : f

...
```

Remarque Dans l'Ex. 5.7, il est demandé de peupler progressivement un `std::vector` de `Person`. Contrairement à ce qu'il en est avec les types numériques rencontrés jusqu'ici, la méthode `push_back()` [36] n'est pas la seule ni nécessairement la plus efficace pour réaliser une insertion en fin de `std::vector`.

S'il s'agit d'insérer un objet *préalablement* existant, alors `push_back()` est la méthode à utiliser :

```
Person lovelace { { "Augusta", "Ada" }, "King-Noel", Sex::FEMALE,
                  { 1815, 12, 10 } };;
 ...
vector<Person> vp;
 ...
vp.push_back(lovelace);
 ...
```

Par contre, si l'objet à insérer dans le `std::vector` est *construit lors de l'insertion*, la méthode `emplace_back()` constitue le meilleur choix. Voici comment procéder :

```
vector<Person> vp;
 ...
vp.emplace_back(vector<string> { "Charles" }, "Babbage", Sex::MALE,
                Date { 1791, 12, 26 });
 ...
```

Les arguments fournis à `emplace_back()` sont exactement ceux à donner au constructeur du type des éléments du `std::vector` qu'on désire utiliser. Ce constructeur est invoqué au sein d'`emplace_back()`. Si des exceptions sont levées, cela se passe donc lors de l'appel d'`emplace_back()`.

La raison pour laquelle `emplace_back()` est un meilleur choix dans ce dernier cas est que cette méthode prévient un clonage ici inutile de l'objet à insérer.

36. http://en.cppreference.com/w/cpp/container/vector/push_back

4.2. Tri

Dans les exercices qui suivent, il va s'agir de trier les `Person` obtenues à l'Ex. 5.7. Cependant, un problème se pose. Soit, ayant constaté que la classe `Person` ne possède aucun accesseur en écriture pour la date de naissance, vous avez imposé que cet attribut est constant, ce qui est une excellente idée, soit vous n'avez pas relevé ce point, mais la date de naissance est de toute façon immuable car les attributs de la classe `Date` vous ont été imposés `const` !

Il est dès lors *impossible* de trier un `std::vector` de `Person` car cela ne peut se faire qu'en modifiant, entre autres, la valeur de la date de naissance des cellules successives du conteneur. 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 `Person`. Le premier élément du `std::vector` de pointeurs pointe sur la première personne du `std::vector` de `Person`, le deuxième pointeur su la deuxième `Person`, etc. jusqu'au dernier pointeur pointant sur la dernière `Person` du `std::vector` de `Person`. À la place de trier le `std::vector` de `Person` — opération impossible —, on trie le `std::vector` de pointeurs. En parcourant ensuite le `std::vector` de pointeurs trié et en déréférençant ceux-ci, le résultat est similaire au tri du `std::vector` de `Person` ! C'est cette approche que nous adoptons.

Ex. 5.8 À la suite de votre code réponse à l'Ex. 5.7, produisez un `std::vector` de pointeurs de `const Person` dont chaque élément pointe sur l'élément de même index du `std::vector` de `Person` construit initialement. Les pointeurs sont de type `const Person *` pour rendre impossible toute modification par inadvertance des `Person` du premier `std::vector`.

L'algorithme `std::transform()` [37] allié à la fonction `std::back_inserter()` [38] peuvent être utiles.

En outre, implémentez une fonction de prototype :

```
void print(const std::vector<const gxxxxx::Person *> & c)
```

qui affiche le résultat des déréférencements du contenu d'un `std::vector` de pointeurs de `const Person`.

Affichez les déréférencements du contenu du `std::vector` de pointeurs.

Avec les mêmes données que celles de l'Ex. 5.7, l'affichage est strictement similaire à celui de cet exercice :

```
Nom                : Bah
Prénom(s)          : David Imran Mohamed Youssef
Date de naissance : 27/09/2014
Sexe               : m
```

37. http://en.cppreference.com/w/cpp/algorithm/transform
38. http://en.cppreference.com/w/cpp/iterator/back_inserter

```
Nom                : Bah
Prénom(s)          : Mohamed Youssef Gabriel Yanis Lucas Adam Amir
Date de naissance : 26/05/2015
Sexe               : m

Nom                : Mertens
Prénom(s)          : Aya Lina Nour
Date de naissance : 26/05/2013
Sexe               : f

...
```

Remarque Lorsque vous utilisez les algorithmes standards, n'hésitez pas à recourir aux expressions lambda [39], en ce compris les expressions lambda généralisées [40] du C++14. Attention, il ne s'agit pas de *fonctions* lambda, mais d'*expressions* lambda, ne fût-ce que parce qu'aucune fonction, mais des objets fonctions se cachent derrière [41] ces expressions.

D'autre part, ne soyez pas impressionnés par le nombre ou la complexité des tris demandés de l'Ex. 5.9 à l'Ex. 5.12. Ces exercices sont incrémentaux. On passe de l'un à son suivant en ajoutant une nouvelle condition, exprimée en deux (ou trois) lignes de code, dans la relation d'ordre entre les éléments du conteneur trié.

Ex. 5.9 Reprenez à la suite de votre code réponse à l'Ex. 5.8. Triez le `std::vector` de `const Person *` dans l'ordre chronologique de date de naissance des personnes. C'est-à-dire que dans le `std::vector` de pointeurs trié, le premier pointeur pointe sur la personne la plus âgée du `std::vector` de l'Ex. 5.7 et le dernier sur la plus jeune.

L'algorithme `std::sort()` [42] est votre ami.

Affichez les déréférencements du contenu du `std::vector` de pointeurs.

Voici un extrait de ce que cela peut donner :

```
Nom                : Nguyen
Prénom(s)          : Amir
Date de naissance : 27/05/2013
Sexe               : m

Nom                : Bah
Prénom(s)          : Anna Aya
Date de naissance : 27/05/2013
```

39. http://www.cprogramming.com/c++11/c++11-lambda-closures.html
40. https://solarianprogrammer.com/2014/08/28/cpp-14-lambda-tutorial/
41. https://stackoverflow.com/a/23499673
42. http://en.cppreference.com/w/cpp/algorithm/sort

```
Sexe              : f

...

Nom               : Martin
Prénom(s)         : Lina
Date de naissance : 31/05/2014
Sexe              : f

Nom               : Jacobs
Prénom(s)         : Amir Gabriel David Mohamed Youssef Imran
Date de naissance : 27/09/2014
Sexe              : m
```

Ex. 5.10 Triez maintenant le `std::vector` de `const Person *` en majeur sur la date de naissance (ordre chronologique) et en mineur sur le nom de famille (ordre lexicographique tenant compte de la casse, c'est-à-dire des majuscules et minuscules).

Les opérateurs de comparaison [43] de `std::string` sont utiles.

Affichez les déréférencements du contenu du `std::vector` de pointeurs trié.

Voici un extrait de ce que cela peut donner :

```
Nom               : Bah
Prénom(s)         : Anna Aya
Date de naissance : 27/05/2013
Sexe              : f

Nom               : Nguyen
Prénom(s)         : Amir
Date de naissance : 27/05/2013
Sexe              : m

...

Nom               : Martin
Prénom(s)         : Lina
Date de naissance : 31/05/2014
Sexe              : f

Nom               : Jacobs
Prénom(s)         : Amir Gabriel David Mohamed Youssef Imran
```

43. http://en.cppreference.com/w/cpp/string/basic_string/operator_cmp

```
Date de naissance : 27/09/2014
Sexe              : m
```

Ex. 5.11 Triez le `std::vector` de `const Person *` en majeur sur la date de naissance (ordre chronologique), en médian sur le nom de famille (ordre lexicographique tenant compte de la casse) et en mineur sur les prénoms (même relation d'ordre que pour le nom, pour chaque prénom).

Les opérateurs de comparaison [44] de `std::vector` conjugués à ceux des `std::string` conviennent bien dans notre cas.

Affichez les déréférencements du contenu du `std::vector` de pointeurs trié.

Ex. 5.12 Triez le `std::vector` de `const Person *` en appliquant les critères suivants, dans cet ordre :

1. les femmes précèdent les hommes ;
2. les aînés précèdent les cadets ;
3. l'ordre lexicographique tenant compte des majuscules et minuscules est appliqué aux noms de famille ;
4. l'ordre lexicographique tenant compte des majuscules et minuscules est appliqué aux listes de prénoms.

Affichez les déréférencements du contenu du `std::vector` de pointeurs trié.

Voici un extrait de ce que cela peut donner en console :

```
Nom               : Janssens
Prénom(s)         : Lina Aya Sara Yasmine Sofia Anna Nour Emma
Date de naissance : 26/05/2013
Sexe              : f

Nom               : Janssens
Prénom(s)         : Yasmine
Date de naissance : 26/05/2013
Sexe              : f

Nom               : Mertens
Prénom(s)         : Anna Lina Sofia
Date de naissance : 26/05/2013
Sexe              : f

...

Nom               : Janssens
Prénom(s)         : Mohamed Rayan Adam
```

44. http://en.cppreference.com/w/cpp/container/vector/operator_cmp

```
Date de naissance : 26/05/2015
Sexe              : m

Nom               : Martin
Prénom(s)         : Yanis Adam Mohamed Gabriel
Date de naissance : 26/05/2015
Sexe              : m

Nom               : Martin
Prénom(s)         : Yanis Mohamed Amir Gabriel Imran Lucas
Date de naissance : 26/05/2015
Sexe              : m

Nom               : Peeters
Prénom(s)         : Gabriel Mohamed Lucas
Date de naissance : 26/05/2015
Sexe              : m

Nom               : Nguyen
Prénom(s)         : Amir Mohamed
Date de naissance : 27/05/2015
Sexe              : m
```

A. Classe `Date`

A.1. Fichier source `main.cpp`

Voici la version partielle à compléter de `main.cpp` :

```cpp
#include <iostream>

#include "date.h"

int main()
{
    using namespace std;
    using namespace nvs;

    // TODO

    cout << today() << endl;
}
```

A.2. Fichier d'en-têtes `date.h`

Voici la version partielle à compléter de `date.h` :

```
/*!
 * \file date.h
 * \brief Définition de la classe nvs::Date et de fonctions utilisant
 *        ce type.
 */
#ifndef DATE_H
#define DATE_H

#include <limits>
#include <array>
#include <string>
#include <ostream>

/*!
 * \mainpage Une classe pour représenter une date
 *
 * Un bon point d'entrée est celui de la documentation de
 * l'espace de nom \ref nvs.
 */

/*!
 * \brief Espace de nom de Nicolas Vansteenkiste.
 */
namespace nvs
{

/*!
 * \brief Classe représentant une date.
 *
 * On ne se soucie pas ici du type de calendrier et de sa
 * pertinence historique. Il n'y a pas d'années minimale ou
 * maximale autre que celles impliquées par la taille du
 * type sous-jacent.
 *
 * Les attributs \ref year_, \ref month_ et \ref day_ sont constants.
 * Les instances de Date sont donc
 * [immuables](http://www.cnrtl.fr/definition/immuable).
 */
class Date
{
  public:
```

```
42
43      /*!
44       * \brief Valeur minimale acceptée pour l'année.
45       */
46      constexpr static int MINIMUM_YEAR { std::numeric_limits<int>::min() };
47
48      /*!
49       * \brief Valeur maximale acceptée pour l'année.
50       */
51      constexpr static int MAXIMUM_YEAR { std::numeric_limits<int>::max() };
52
53      /*!
54       * \brief Valeur minimale acceptée pour le mois.
55       */
56      constexpr static unsigned MINIMUM_MONTH { 1 };
57
58       /*!
59        * \brief Valeur maximale acceptée pour le mois.
60        */
61      constexpr static unsigned MAXIMUM_MONTH { 12 };
62
63      /*!
64       * \brief Valeur minimale acceptée pour le jour.
65       */
66      constexpr static unsigned MINIMUM_DAY { 1 };
67
68      /*!
69       * \brief Nombre de jours par mois, _hors_ années bissextiles.
70       *
71       * Il s'agit des valeurs maximales acceptées pour le jour
72       * en fonction du mois, _sans_ tenir compte des années
73       * bissextiles. Pour connaître le nombre de jours d'un mois
74       * en tenant compte des années bissextiles, il faut utiliser
75       * la méthode dayNumberInMonth().
76       *
77       * La valeur initiale nulle est présente pour permettre
78       * d'utiliser directement la valeur du mois comme index
79       * de la `std::array`.
80       */
81      constexpr static std::array < unsigned,
82                MAXIMUM_MONTH + 1 > MAXIMUM_DAY
83      {{ 0, 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 }};
84
85    private:
```

```
86
87      /*!
88       * \brief L'année.
89       *
90       * Cet attribut est constant.
91       *
92       * Elle est comprise entre \ref MINIMUM_YEAR et
93       * \ref MAXIMUM_YEAR, ces valeurs comprises.
94       *
95       * On considère qu'il y a une
96       * [année 0](https://fr.wikipedia.org/wiki/Ann%C3%A9e_z%C3%A9ro).
97       */
98      const int year_;
99
100     /*!
101      * \brief Le mois.
102      *
103      * Cet attribut est constant.
104      *
105      * Janvier a comme valeur \ref MINIMUM_MONTH, février
106      * \ref MINIMUM_MONTH + 1, etc., jusquà décembre de valeur
107      * \ref MAXIMUM_MONTH.
108      */
109     const unsigned month_;
110
111     /*!
112      * \brief Le jour.
113      *
114      * Cet attribut est constant.
115      *
116      * Le premier jour du mois a comme valeur \ref MINIMUM_DAY, le
117      * suivant \ref MINIMUM_DAY + 1, etc., jusqu'au dernier du mois,
118      * dépendant du mois et de l'année et fourni par
119      * dayNumberInMonth().
120      */
121     const unsigned day_;
122
123   public:
124
125     /*!
126      * \brief Constructeur.
127      *
128      * Ce constructeur valide ses arguments à l'aide du modèle
129      * de fonction validate(). Dès lors,
```

```
130      * une exception de type ‘std::invalid_argument‘ peut être levée
131      * lors de son utilisation.
132      *
133      * On considère qu’il y a une
134      * [année 0](https://fr.wikipedia.org/wiki/Ann%C3%A9e_z%C3%A9ro).
135      *
136      * \param year l’année.
137      * \param month le mois.
138      * \param day le jour.
139      *
140      * \throw std::invalid_argument si :
141      *  + year \f$\notin\f$ [Date::MINIMUM_YEAR, Date::MAXIMUM_YEAR]
142      *  + month \f$\notin\f$ [Date::MINIMUM_MONTH, Date::MAXIMUM_MONTH]
143      *  + day \f$\notin\f$ [Date::MINIMUM_DAY, Date::MAXIMUM_DAY[month]]
144      *  tenant compte des années bissextiles.
145      *
146      * \see year_, month_, day_.
147      */
148     Date(int year, unsigned month, unsigned day);
149
150     /*!
151      * \brief Accesseur en lecture de l’année.
152      *
153      * \return l’année.
154      */
155     inline int year() const;
156
157     /*!
158      * \brief Accesseur en lecture du mois.
159      *
160      * \return le mois.
161      */
162     inline unsigned month() const;
163
164     /*!
165      * \brief Accesseur en lecture du jour.
166      *
167      * \return le jour.
168      */
169     inline unsigned day() const;
170
171     /*!
172      * \brief Méthode pour déterminer si l’année courante est
173      *        bissextile.
```

```
174         *
175         * \return ‘true‘ si l’année courante est bissextile, ‘false‘
176         *         sinon.
177         *
178         * \see leapYear(int).
179         */
180        inline bool leapYear() const;
181
182        /*!
183         * \brief Méthode pour connaître le nombre de jours du mois
184         *        courant.
185         *
186         * \return le nombre de jours du mois courant.
187         */
188        inline unsigned dayNumberInMonth() const;
189
190        /*!
191         * \brief Méthode pour obtenir la date courante sous la forme
192         *        d’une std::string.
193         *
194         * La ‘std::string‘ retournée est ‘"jj/mm/aaaa"‘, par exemple
195         * ‘"26/04/2016"‘.
196         *
197         * Il s’agit de la même forme que celle lors de l’injection
198         * d’une Date dans un flux en sortie
199         * (voir operator<<(std::ostream &, const Date &)).
200         *
201         * \return une représentation de la date courante sous la forme
202         *         d’une std::string.
203         *
204         * \see to_string(const Date &).
205         */
206        std::string to_string() const;
207    };
208
209    // prototypes
210
211    /*!
212     * \brief Opérateur d’injection d’une Date dans un flux en sortie.
213     *
214     * La Date est injectée sous la forme ‘jj/mm/aaaa‘, par exemple
215     * ‘26/04/2016‘.
216     *
217     * Il s’agit de la même forme que celle
```

```
218  * retournée par \ref nvs::Date::to_string().
219  *
220  * \param out le flux dans lequel la Date est injectée.
221  * \param in la Date injectée dans le flux en sortie.
222  *
223  * \return le flux dans lequel la Date a été injectée.
224  */
225 std::ostream & operator<<(std::ostream & out, const Date & in);
226 
227 /*!
228  * \brief Opérateur de comparaison de deux dates.
229  *
230  * La valeur `true` est retournée si `lhs` est
231  * _strictement antérieure_ à `rhs`. Lorsque `lhs` est
232  * _égale ou postérieure_ à `rhs`, `false` est retourné.
233  *
234  * \param lhs une Date.
235  * \param rhs une autre Date.
236  * \return `true` si lhs est strictement antérieure à rhs,
237  *         `false` sinon.
238  */
239 inline bool operator<(const Date & lhs, const Date & rhs);
240 
241 /*!
242  * \brief Opérateur de test d'égalité de deux dates.
243  *
244  * \param lhs une Date.
245  * \param rhs une autre Date.
246  * \return `true` si les deux dates sont identiques, `false` sinon.
247  */
248 inline bool operator==(const Date & lhs, const Date & rhs);
249 
250 /*!
251  * \brief Fonction de conversion d'une nvs::Date en std::string.
252  *
253  * La `std::string` retournée est `"jj/mm/aaaa"`, par exemple
254  * `"26/04/2016"`.
255  *
256  * Il s'agit de la même forme que celle lors de l'injection
257  * d'une Date dans un flux en sortie
258  * (voir operator<<(std::ostream &, const Date &)).
259  *
260  * L'appel :
261  *
```

```
262  *      to_string(in);
263  *
264  * est équivalent à celui-ci :
265  *
266  *      in.to_string();
267  *
268  * \param in la Date à convertir.
269  *
270  * \return une représentation de la date courante sous la forme
271  *         d'une std::string.
272  *
273  * \see Date::to_string().
274  */
275 inline std::string to_string(const Date & in);
276
277 /*!
278  * \brief Fonction retournant la date actuelle (au moment de
279  *        l'exécution) sous la forme d'une Date.
280  *
281  * \return la date au moment de l'exécution comme une Date.
282  *
283  * \see http://en.cppreference.com/w/cpp/chrono/time_point
284  * \see http://en.cppreference.com/w/cpp/chrono/c/time
285  * \see http://en.cppreference.com/w/cpp/chrono/c/localtime
286  */
287 Date today();
288
289 /*!
290  * \brief Fonction pour déterminer si une année est bissextile.
291  *
292  * \param year l'année à tester.
293  *
294  * \return `true` si `year` est bissextile, `false` sinon.
295  *
296  * \see Date::leapYear().
297  */
298 inline bool leapYear(int year);
299
300 // implémentations inline
301
302 // fonctions inline
303
304 bool operator<(const Date & lhs, const Date & rhs)
305 {
```

```
306     // TODO
307     return false;
308 }
309
310 bool operator==(const Date & lhs, const Date & rhs)
311 {
312     // TODO
313     return false;
314 }
315
316 std::string to_string(const Date & in)
317 {
318     // TODO
319     return "·.·";
320 }
321
322 bool leapYear(int year)
323 {
324     // TODO
325     return false;
326 }
327
328 // méthodes inline
329
330 int Date::year() const
331 {
332     // TODO
333     return 0;
334 }
335
336 unsigned Date::month() const
337 {
338     // TODO
339     return 0;
340 }
341
342 unsigned Date::day() const
343 {
344     // TODO
345     return 0;
346 }
347
348 bool Date::leapYear() const
349 {
```

```
350     // TODO
351     return false;
352 }
353
354 unsigned Date::dayNumberInMonth() const
355 {
356     // TODO
357     return 0;
358 }
359
360 } // namespace nvs
361
362 #endif // DATE_H
```

A.3. Fichier source `date.cpp`

Voici la version partielle à compléter de `date.cpp` :

```
1  #include "date.h"
2
3  #include <array>         // pour std::array
4  #include <string>        // pour std::string
5  #include <ostream>       // pour std::ostream
6  #include <chrono>        // pour std::chrono, std::chrono::time_point
7  //                               std::chrono::system_clock,
8  //                               std::chrono::system_clock::now,
9  //                               std::chrono::system_clock::to_time_t
10 #include <ctime>         // pour std::time_t, std::tm, std::localtime,
11 //                               std::time
12
13 // TODO : éventuellement ajouter des #include
14
15 namespace nvs
16 {
17
18 // définition des attributs statiques
19
20 constexpr int Date::MINIMUM_YEAR;
21 constexpr int Date::MAXIMUM_YEAR;
22 constexpr unsigned Date::MINIMUM_MONTH;
23 constexpr unsigned Date::MAXIMUM_MONTH;
24 constexpr unsigned Date::MINIMUM_DAY;
25 constexpr std::array<unsigned, Date::MAXIMUM_MONTH + 1>
26                                                 Date::MAXIMUM_DAY;
```

```
27
28  // méthodes
29
30  Date::Date(int year, unsigned mounth, unsigned day) :
31      // TODO
32      year_ { 0 },
33      month_ { 0 },
34      day_ { 0 }
35  { }
36
37  std::string Date::to_string() const
38  {
39      // TODO
40      return "·.·";
41  }
42
43  // méthodes statiques
44
45  // fonctions
46
47  std::ostream & operator<<(std::ostream & out, const Date & in)
48  {
49      // TODO
50      return out << "·.·";;
51  }
52
53  Date today()
54  {
55      std::chrono::time_point<std::chrono::system_clock> now
56      { std::chrono::system_clock::now() };
57      std::time_t ttnow { std::chrono::system_clock::to_time_t(now) };
58      /*
59      std::time_t ttnow = std::time(nullptr);
60      */ // old c-style
61      std::tm * nowUsefull = std::localtime(&ttnow);
62
63      return Date{ nowUsefull->tm_year + 1900,
64                   static_cast<unsigned>(nowUsefull->tm_mon) + 1,
65                   static_cast<unsigned>(nowUsefull->tm_mday) };
66  }
67
68  } // namespace nvs
```

B. Fonctions de validation

```cpp
/*!
 * \file validation.hpp
 * \brief Fonctions d'aide à la validation.
 */
#ifndef VALIDATION_HPP
#define VALIDATION_HPP

#include <functional>
#include <utility>
#include <string>
#include <stdexcept>

namespace nvs
{

/*!
 * \brief Évaluation si une valeur est comprise entre deux autres.
 *
 * Les opérateurs `<` et `==` doivent être disponibles pour le type
 * `T`.
 *
 * \param value valeur à tester.
 * \param min borne minimale.
 * \param max borne maximale.
 *
 * \return `true` si `min <= value <= max`, `false` sinon.
 */
template<typename T>
bool between(const T & value, T min, T max)
{
    // reference_wrapper pour que ça fonction avec classe immuable
    std::reference_wrapper<T> min_ { min };
    std::reference_wrapper<T> max_ { max };

    if (max_ < min_) std::swap(min_, max_);

    return (min_ < value || min_ == value) &&
           (value < max_ || value == max_);
}

/*!
 * \brief Production d'un message d'erreur.
```

```
43  *
44  * La fonction `to_string()` doit être disponible pour le type `T`.
45  *
46  * \param value valeur.
47  * \param min borne minimale.
48  * \param max borne maximale.
49  * \param msg en-tête du message.
50  *
51  * \return message d'erreur.
52  *
53  * \see validate()
54  */
55 template<typename T>
56 std::string error_message(const T & value, T min, T max,
57                           const std::string & msg = "error : ")
58 {
59     // pour les types primitifs hors namespace nvs...
60     using std::to_string;
61
62     // reference_wrapper pour que ça fonction avec classe immuable
63     std::reference_wrapper<T> min_ { min };
64     std::reference_wrapper<T> max_ { max };
65
66     if (max_ < min_) std::swap(min_, max_);
67
68     return msg + to_string(value) + " not in [" +
69            to_string(min_) + ", " + to_string(max_) + "]";
70 }
71
72 /*!
73  * \brief Validation d'une valeur entre deux bornes.
74  *
75  * \param value valeur à valider.
76  * \param min borne minimale.
77  * \param max borne maximale.
78  * \param msg en-tête du message en cas de problème.
79  *
80  * \return `value` si `min <= value <= max`.
81  *
82  * \throw std::invalid_argument si
83  *              value \f$\notin\f$ [`min`, `max`]
84  *
85  * \see between()
86  */
```

```
87  template<typename T>
88  T validate(const T & value, const T & min, const T & max,
89             const std::string & msg = "error : ")
90  {
91      if (!between(value, min, max))
92      {
93          throw std::invalid_argument { error_message(value, min,
94                                                      max, msg) };
95      }
96
97      return value;
98  }
99
100 } // namespace nvs
101
102 #endif // VALIDATION_HPP
```

C. Énumération fortement typée

enum class FormOption

```
1   /*!
2    * \file formoption.h
3    * \brief Définition de l'énumération fortement typée
4    *        nvs::FormOption.
5    */
6   #ifndef FORMOPTION_H
7   #define FORMOPTION_H
8
9   namespace nvs
10  {
11
12  /*!
13   * \brief Énumération pour spécifier les options de mise en forme
14   *        d'impression ou de conversion binaire → chaîne de
15   *        caractères.
16   */
17  enum class FormOption
18  {
19      /*!
20       * \brief Absence de spécification.
21       */
22      NONE = 0b0000,
```

```
23      // binary literals : c++14 :
24      // http://www.informit.com/articles/article.aspx?p=2209021
25
26      /*!
27       * \brief Forme courte.
28       */
29      SHORT = 0b0001,
30
31      /*!
32       * \brief Forme longue.
33       */
34      LONG = 0b0010
35  };
36
37  } // namespace nvs
38
39  #endif // FORMOPTION_H
```

D. Fonctions de tests des classes `class Date` et `class Person`

```
1   /*!
2    * \file testgenerator.h
3    * \brief Définition de fonctions pour la création de Date et Person.
4    */
5   #ifndef TESTGENERATOR_H
6   #define TESTGENERATOR_H
7
8   #include <tuple>
9   #include <string>
10  #include <vector>
11
12  /*!
13   * \mainpage Des fonctions pour tester les classes Date et Person
14   *
15   * Un bon point d'entrée est celui de la documentation de
16   * l'espace de nom \ref nvs.
17   *
18   */
19
20  /*!
21   * \brief Espace de nom de Nicolas Vansteenkiste.
```

```
22  */
23 namespace nvs
24 {
25
26 /*!
27  * \brief Générateur d'arguments pour la construction d'une Date.
28  *
29  * Parmi les valeurs retournées, certaines permettent de produire
30  * une Date correcte, d'autres devraient produire une erreur lors
31  * de la création de la Date correspondante.
32  *
33  * Le générateur de nombres uniformément aléatoires urng() est
34  * utilisé, sans bruit. Si vous désirez des séquences différentes
35  * d'une exécution à l'autre, il faut invoquer randomize() _avant_
36  * dategenerator().
37  *
38  * \return un std::tuple dont :
39  *           * le premier élément (celui d'indice 0) est à utiliser
40  *             comme argument pour l'_année_ ;
41  *           * le deuxième élément (celui d'indice 1) est à utiliser
42  *             comme argument pour le _mois_ ;
43  *           * le dernier élément (celui d'indice 2) est à utiliser
44  *             comme argument pour le _jour_.
45  */
46 std::tuple<int, unsigned, unsigned> dategenerator();
47
48 /*!
49  * \brief Générateur d'arguments pour la construction d'une Person.
50  *
51  * Cette fonction retourne l'ensemble des arguments nécessaires
52  * à la création d'une instance de Person.
53  * Aucun des ensembles de valeurs retournés ne produit une erreur
54  * lors de son utilisation par les constructeurs de Person tels que
55  * ceux demandés, mais :
56  *  + dans la liste des prénoms, certains peuvent apparaître
57  *    plusieurs fois ;
58  *  + les listes de prénoms sont de longueurs variables ;
59  *  + il n'y a que deux valeurs possibles pour le sexe de la
60  *    Person :
61  *      - un ''f'' indique une femme ;
62  *      - un ''m'' indique une homme.
63  *
64  * Le générateur de nombres uniformément aléatoires urng() est
65  * utilisé, sans bruit. Si vous désirez des séquences différentes
```

```
 * d'une exécution à l'autre, il faut invoquer randomize() _avant_
 * people().
 *
 * \param size le nombre d'éléments du std::vector retourné.
 *
 * \return un std::vector de std::tuple dont :
 *          + le premier élément (celui d'indice 0) est un
 *            std::vector de std::string à utiliser
 *            comme argument pour la _liste de prénoms_ de la
 *            Person ;
 *          + le deuxième élément (celui d'indice 1) est une
 *            std::string à utiliser
 *            comme argument pour le _nom_ de la Person ;
 *          + le troisième élément (celui d'indice 2) est un char à
 *            utiliser pour déterminer le _sexe_ de la Person :
 *              - ''f'' pour `Sex::FEMALE` ;
 *              - ''m'' pour `Sex::MALE` ;
 *          + le dernier élément (celui d'indice 3) est un
 *            std::tuple à utiliser pour fixer la
 *            _date de naissance_ de la Person en prenant :
 *              - en guise d'_année de naissance_ le premier
 *                élément de ce std::tuple, c'est-à-dire celui
 *                d'index 0 ;
 *              - en guise de _mois de naissance_ le deuxième
 *                élément de ce std::tuple, c'est-à-dire celui
 *                d'index 1 ;
 *              * en guise de _jour de naissance_ le deuxième
 *                élément de ce std::tuple, c'est-à-dire celui
 *                d'index 2.
 */
std::vector<std::tuple<std::vector<std::string>, std::string, char,
    std::tuple<int, unsigned, unsigned>>>
    people(unsigned size = 1000);

} // namespace nvs

#endif // TESTGENERATOR_H
```

```
#include "testgenerator.h"
#include "../random/random.hpp"

#include <tuple>
#include <array>
#include <vector>
```

```cpp
#include <string>
#include <algorithm>

using namespace std;

namespace nvs
{

tuple<int, unsigned, unsigned> dategenerator()
{
    static constexpr array<array<unsigned, 6>, 3> data {{
            { 2000, 1900, 0, 10000, 2014, 2020 },
            { 0, 4, 2, 12, 13, 2 },
            { 0, 102, 28, 29, 30, 31 }
        }};    // double accolade car aggrégat

    return tuple<int, unsigned, unsigned>
    {
        data[0][random_value(0, static_cast<int>(data[0].size()) - 1)],
        data[1][random_value(0, static_cast<int>(data[1].size()) - 1)],
        data[2][random_value(0, static_cast<int>(data[2].size()) - 1)]
    };
}

vector<tuple<vector<string>, string, char,
       tuple<int, unsigned, unsigned>>> people(unsigned size)
{
    // http://statbel.fgov.be/fr/statistiques/chiffres/population/noms/

    static const array<array<string, 10>, 2> possibleFN {{
            {
                "Lina", "Aya", "Sarah", "Sofia", "Nour", "Yasmine",
                "Malak", "Emma", "Sara", "Anna"
            },
            {
                "Adam", "Mohamed", "Rayan", "Gabriel", "David",
                "Imran", "Amir", "Lucas", "Youssef", "Yanis"
            }
        }};

    static constexpr array<char, 2> possibleS {'f', 'm'};

    static const array<string, 10> possibleN
    {
```

```
51          "Diallo", "Bah", "Janssens", "Peeters", "Dubois",
52          "Nguyen", "Barry", "Jacobs", "Mertens", "Martin" };
53
54      // make_tuple : constexpr c++14
55      static constexpr array<tuple<int, unsigned, unsigned>, 10>
56      possibleB
57      {
58          make_tuple(2014, 5u, 27u), make_tuple(2014, 5u, 20u),
59          make_tuple(2014, 9u, 27u), make_tuple(2013, 5u, 27u),
60          make_tuple(2015, 5u, 27u), make_tuple(2013, 5u, 28u),
61          make_tuple(2013, 5u, 26u), make_tuple(2013, 6u, 27u),
62          make_tuple(2014, 5u, 31u), make_tuple(2015, 5u, 26u)
63      };
64
65      vector<tuple<vector<string>, string, char,
66             tuple<int, unsigned, unsigned>>> ret;
67
68      for (unsigned u {0}; u < size; ++u)
69      {
70          unsigned indexS { random_value(0u, static_cast<unsigned>(
71                                         possibleS.size()) - 1) };
72          unsigned indexN { random_value(0u, static_cast<unsigned>(
73                                         possibleN.size()) - 1) };
74          unsigned indexB { random_value(0u, static_cast<unsigned>(
75                                         possibleB.size()) - 1) };
76          vector<string> fn(random_value(1u, 9u));
77          generate(begin(fn), end(fn), [indexS]()
78          {
79              return possibleFN[indexS]
80                     [nvs::random_value(0u,
81                                        static_cast<unsigned>(
82                                            possibleFN[indexS].size())
83                                        - 1)];
84              // Q : pq pas besoin de capturer possibleFN ?
85              // car possibleFN est static ? const ?
86          });
87          ret.emplace_back(move(fn), possibleN[indexN],
88                           possibleS[indexS],
89                           possibleB[indexB]);
90      }
91
92      return ret;
93  }
94
```

```
95 } // namespace nvs
```

E. Génération de nombres (pseudo-)aléatoires

```
1  /*!
2   * \file random.hpp
3   * \brief Définitions de fonctions conviviales pour générer des
4   *        séquences pseudo-aléatoires.
5   */
6  #ifndef RANDOM_HPP
7  #define RANDOM_HPP
8
9  #include <random>
10 #include <utility>
11 #include <limits>
12 #ifdef _WIN32
13 #include <ctime>
14 #endif
15
16 /*!
17  * \brief Espace de nom de Nicolas Vansteenkiste.
18  *
19  */
20 namespace nvs
21 {
22
23 // fonctions
24
25 /*!
26  * \brief Un générateur de nombres uniformément aléatoires.
27  *
28  * Cette fonction produit et partage un unique
29  * générateur de nombres uniformément aléatoires
30  * (_Uniform Random Number Generator_).
31  * Elle est issue de Random Number Generation in C++11
32  * ([WG21 N3551]
33  * (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf)),
34  * par Walter E. Brown.
35  *
36  * _Remarque_ : Sous Windows, c'est un std::mt19937 qui est
37  * retourné, sous les autres systèmes d'exploitation c'est
38  * un std::default_random_engine. La raison en est qu'avec
```

```
39  * gcc sous Windows, la première valeur retournée par
40  * un std::default_random_engine change peu en fonction de la
41  * graine plantée avec nvs::randomize. Pour s'en convaincre,
42  * exécuter nvs::random_value(1, 100000) par des instances
43  * successives d'un même programme...
44  *
45  * \return un générateur de nombres uniformément aléatoires.
46  */
47 inline auto & urng()
48 {
49 #ifdef _WIN32
50     static std::mt19937 u {};
51     // https://stackoverflow.com/a/32731387
52     // dans le lien précédent : Linux   <-> gcc
53     //                          et Windows <-> msvc
54 #else
55     static std::default_random_engine u {};
56 #endif
57     return u;
58 }
59
60 /*!
61  * \brief Un peu de bruit.
62  *
63  * Cette fonction met le générateur de nombres uniformément
64  * aléatoires partagé par nvs::urng() dans un état aléatoire.
65  * Elle est issue de Random Number Generation in C++11
66  * ([WG21 N3551]
67  * (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf)),
68  * par Walter E. Brown.
69  */
70 inline void randomize()
71 {
72 #ifdef _WIN32
73     urng().seed(std::time(nullptr));
74     // https://stackoverflow.com/a/18908041
75 #else
76     static std::random_device rd {};
77     urng().seed(rd());
78 #endif
79 }
80
81 /*!
82  * \brief Générateur de flottants aléatoires.
```

```
 *
 * Les flottants produits se distribuent uniformément entre
 * ‘min‘ et ‘max‘, la valeur minimale comprise, la maximale non.
 *
 * Si ‘max‘ est strictement inférieur à ‘min‘, les contenus de ces
 * variables sont permutés.
 *
 * Cette fonction est largement inspirée par Random Number
 * Generation in C++11 ([WG21 N3551]
 * (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf)),
 * par Walter E. Brown.
 *
 * _Remarque_ : Par rapport au modèle de fonction
 * nvs::random_value<T> produisant des entiers aléatoires, les
 * arguments ‘min‘ et ‘max‘ sont inversés de sorte à avoir la
 * valeur nulle (0) comme borne (minimale ou maximale) si la
 * fonction est appelée avec un seul argument. Notez que cela n'a
 * pas de réelle incidence sur la signification des paramètres
 * puisque leurs contenus sont permutés si nécessaire.
 *
 * \param max borne supérieure (ou inférieure) de l'intervalle
 *            dans lequel les flottants sont générés.
 * \param min borne inférieure (ou supérieure) de l'intervalle
 *            dans lequel les flottants sont générés.
 *
 * \return un flottant dans l'intervalle semi-ouvert à droite
 *         [‘min‘, ‘max‘[ (ou [‘max‘, ‘min‘[ si ‘max‘ < ‘min‘).
 */
inline double random_value(double max = 1., double min = 0.)
{
    static std::uniform_real_distribution<double> d {};

    if (max < min) std::swap(min, max);

    return d(urng(), decltype(d)::param_type {min, max});
}

// fonctions template

/*!
 * \brief Générateur d'entiers aléatoires.
 *
 * Les entiers produits se distribuent uniformément entre
 * ‘min‘ et ‘max‘, ces valeurs incluses.
```

```
127  *
128  * The effect is undefined if T is not one of : short, int, long,
129  * long long, unsigned short, unsigned int, unsigned long, or
130  * unsigned long long.
131  *
132  * Si ‘max‘ est strictement inférieur à ‘min‘, les contenus de ces
133  * variables sont permutés.
134  *
135  * Cette fonction est largement inspirée par Random Number
136  * Generation in C++11 ([WG21 N3551]
137  * (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf)),
138  * par Walter E. Brown.
139  *
140  * \param min valeur minimale (ou maximale) pouvant être retournée.
141  * \param max valeur maximale (ou minimale) pouvant être retournée.
142  *
143  * \return un entier entre ‘min‘ et ‘max‘.
144  */
145 template<typename T = int>
146 inline T random_value(T min = std::numeric_limits<T>::min(),
147                       T max = std::numeric_limits<T>::max())
148 {
149     static std::uniform_int_distribution<T> d {};
150
151     if (max < min) std::swap(min, max);
152
153     return d(urng(), typename decltype(d)::param_type {min, max});
154 }
155
156 } // namespace nvs
157
158 #endif // RANDOM_HPP
```