CPPLI : TD 9 : C++ : Composition et héritage

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

Année académique 2019 – 2020

Ce TD [1] poursuit la création de classes [2] en C++ via la composition [3] et l'héritage [4].

Réalisez ce TD en exploitant le standard C++17 [5].

1. Présentation

Au long de ce TD, vous êtes amené à coder un ensemble de classes qui représentent un jeu simplifié de lotto [6].

On a des valeurs entières positives $n \in \mathbb{N}$ telles que :

$$v_{min} \leqslant n \leqslant v_{max},$$

*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/td09_cpp/td09_cpp_withAppendix.pdf (consulté le 3 décembre 2019).
2. https://en.cppreference.com/w/cpp/language/class (consulté le 3 décembre 2019).
3. https://fr.wikipedia.org/wiki/Composition_(programmation) (consulté le 3 décembre 2019).
4. https://en.cppreference.com/w/cpp/language/derived_class (consulté le 3 décembre 2019).
5. https://isocpp.org/search/google?q=c%2B%2B17 (consulté le 27 novembre 2019).
6. https://fr.wikipedia.org/wiki/Loto (consulté le 3 décembre 2019).

où v_{min}, $v_{max} \in \mathbb{N}$ et $v_{min} \leqslant v_{max}$.

Parmi ces $v_{max} - v_{min} + 1$ valeurs, on en sélectionne aléatoirement N.

Le but du jeu est de prédire ces N valeurs ou, à défaut, au moins 1 ou 2 etc. ou $N - 1$ parmi celles-ci.

2. Classe `Parameter`

Le fichier `parameter.h`[7] contient la définition de la `class Parameter`. Cette classe permet le paramétrage du jeu de lotto, c'est-à-dire la définition des valeurs v_{min}, v_{max} et N mentionnées dans la présentation (section 1).

La documentation au format html de `class Parameter` est disponible après décompression du fichier `html.7z`[8].

Ce fichier d'en-têtes est reproduit en annexe A, mais référez-vous plutôt à sa documentation html.

Ex. 9.1 Lisez la documentation du fichier `parameter.h`.

Ex. 9.2 Placez la classe `Parameter` dans l'espace de nom `gxxxxx::lotto`, où `gxxxxx` est votre identifiant à l'ESI.

3. Classe `Item`

Le fichier `item_incomplete.h`[9] contient la définition légèrement incomplète de la `class Item`. Cette classe est une classe non instanciable qui sert de base :

— à la classe `Draw` qui représente un tirage aléatoire de valeurs (voir section 4) ;

— à la classe `Pronostic` qui modélise un pronostic de valeurs (voir section 5).

Pour le paramétrage du jeu, elle utilise la classe `Parameter` (voir section 2).

Sa documentation au format html est disponible après décompression du fichier `html.7z`.

Le fichier `item_incomplete.h` est reproduit en annexe B, mais référez-vous plutôt à sa documentation html.

Ex. 9.3 Lisez la documentation de la classe `Item`.

Ex. 9.4 Prenez le fichier `item_incomplete.h` et :

1. renommez-le en `item.h` ;

7. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/parameter.h (consulté le 3 décembre 2019).
8. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/html.7z (consulté le 3 décembre 2019).
9. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/item_incomplete.h (consulté le 3 décembre 2019).

2. adaptez là où nécessaire son contenu à son nouveau nom ;
3. déplacez la classe `Item` et ses fonctions dans l'espace de noms (ou de nommage) `gxxxxx::lotto` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez le fichier `item.h` aux endroits marqués d'un commentaire `// TODO` de sorte que son contenu respecte les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

4. Classe `Draw`

Le fichier `draw.h` [10] contient la définition de la `class Draw`. Cette classe dérive de la classe `Item` (voir section 3). Elle représente un tirage aléatoire de valeurs. Le but du jeu est de pronostiquer ces valeurs.

Sa documentation au format html est disponible après décompression du fichier `html.7z`.

Le fichier `draw.h` est reproduit en annexe C, mais référez-vous plutôt à sa documentation html. Pour la génération de valeurs aléatoires, `draw.h` inclut le fichier `random.hpp` [11] reproduit en annexe D.

Ex. 9.5 Lisez la documentation du fichier `draw.h`.

Ex. 9.6 Placez la classe `Draw` dans l'espace de nom `gxxxxx::lotto`, où `gxxxxx` est votre identifiant à l'ESI.

5. Classe `Pronostic`

Le fichier `pronostic_incomplete.h` [12] contient la définition presque complète de la `class Pronostic`. Cette classe dérive de la classe `Item` (voir section 3). Elle représente un pronostic visant à prédire les valeurs d'un tirage aléatoire.

Sa documentation au format html est disponible après décompression du fichier `html.7z`.

Le fichier `pronostic_incomplete.h` est reproduit en annexe E, mais référez-vous plutôt à sa documentation html.

Ex. 9.7 Lisez la documentation de la classe `Pronostic`.

10. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/draw.h (consulté le 3 décembre 2019).
11. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/random/random.hpp (consulté le 3 décembre 2019).
12. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/pronostic_incomplete.h (consulté le 3 décembre 2019).

Ex. 9.8 Prenez le fichier `pronostic_incomplete.h` et :

1. renommez-le en `pronostic.h` ;
2. adaptez là où nécessaire son contenu à son nouveau nom ;
3. déplacez la classe `Pronostic` dans l'espace de noms `gxxxxx::lotto` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez le fichier `pronostic.h` à l'endroit marqué d'un commentaire `// TODO` de sorte que son contenu respecte les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

6. Classe `Lotto`

Le fichier `lotto_incomplete.h`[13] contient la définition largement incomplète de la `class Lotto`. Cette classe implémente un jeu de lotto. Elle est composée :

— d'une instance de la classe `Parameter` pour son paramétrage ;

— d'une collection d'instances de `Pronostic` pour les pronostics ;

— d'une instance optionnelle[14] de `Draw` pour le tirage au sort.

Ses méthodes veillent, bien entendu, à ce qu'il soit impossible de tricher.

Sa documentation au format html est disponible après décompression du fichier `html.7z`.

Le fichier `lotto_incomplete.h` est reproduit en annexe F, mais référez-vous plutôt à la documentation de la classe `Lotto` et de ses fonctions.

Ex. 9.9 Lisez la documentation de la classe `Lotto` et de ses fonctions.

Ex. 9.10 Prenez le fichier `lotto_incomplete.h` et :

1. renommez-le en `lotto.h` ;
2. adaptez là où nécessaire son contenu à son nouveau nom ;
3. déplacez la classe `Lotto` et ses fonctions dans l'espace de noms (ou de nommage) `gxxxxx::lotto` où `xxxxx` est votre numéro d'étudiant ;
4. complétez et modifiez le fichier `pronostic.h` aux endroits marqués d'un commentaire `// TODO` de sorte que son contenu respecte les spécifications renseignées dans la documentation ;
5. testez exhaustivement !

13. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/lotto_incomplete.h (consulté le 3 décembre 2019).
14. https://en.cppreference.com/w/cpp/utility/optional (consulté le 3 décembre 2019).

Pour vos tests, utilisez le générateur de données de pronostics `data(...)` déclaré dans le fichier `data.h`[15] et implémenté dans le fichier `data.cpp`[16]. Ces fichiers sont reproduits en annexe G.1 et G.2, respectivement.

A. Classe `Parameter`

```
1  /*!
2   * \file parameter.h
3   *
4   * \brief Définition de la classe nvs::lotto::Parameter ainsi que
5   *        de fonctions associées.
6   */
7  #ifndef PARAMETER_H
8  #define PARAMETER_H
9
10 #include <string>
11 #include <ostream>
12 #include <stdexcept>
13
14 // à partir de c++17, il est plus aisé qu'avant de définir des
15 // espaces de nom imbriqués :
16 // http://www.nuonsoft.com/blog/2017/08/01/c17-nested-namespaces/
17 /*!
18  * \brief Espace de nom du projet Lotto de Nicolas Vansteenkiste.
19  */
20 namespace nvs::lotto
21 {
22
23 /*!
24  * \brief Classe de paramétrage d'un jeu de lotto.
25  *
26  * La grille du jeu de lotto est caractérisée par
27  * trois paramètres :
28  *
29  *    + la valeur minimale entière positive des valeurs à prédire ;
30  *    + la valeur maximale entière positive des valeurs à prédire ;
31  *    + le nombre maximal de valeurs à prédire.
32  *
```

15. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/data/data.h (consulté le 3 décembre 2019).
16. https://poesi.esi-bru.be/pluginfile.php/1320/mod_folder/content/0/td09_cpp/ressource/data/data.cpp (consulté le 3 décembre 2019).

```
33  * Les valeurs à prédire sont à choisir parmi tous les entiers
34  * compris entre les valeurs minimale et maximale, ces valeurs
35  * incluses.
36  *
37  * Par exemple, avec :
38  *
39  *    + valeur minimale égale à 6 ;
40  *    + valeur maximale égale à 21 ;
41  *    + nombre de valeurs à prédire égal à 4 ;
42  *
43  * il faut prédire 4 valeurs parmi les entiers entre 6 et 21, ces
44  * valeurs comprises.
45  */
46 class Parameter
47 {
48   public:
49
50     // à partir de c++17, les attributs static peuvent être inline
51     // par ailleurs constexpr implique inline dans ce cas.
52     // voir les liens suivants :
53     // https://stackoverflow.com/a/39653928
54     // http://www.nuonsoft.com/blog/2017/08/28/c17-inline-variables/
55     /*!
56      * \brief Valeur minimale par défaut des chiffres à prédire.
57      */
58     static constexpr unsigned MINIMUM_DEFAULT_ { 1 };
59
60     /*!
61      * \brief Valeur maximale par défaut des chiffres à prédire.
62      */
63     static constexpr unsigned MAXIMUM_DEFAULT_ { 50 };
64
65     /*!
66      * \brief Valeur par défaut du nombre de chiffres à prédire.
67      */
68     static constexpr unsigned LENGTH_DEFAULT_ { 8 };
69
70   private:
71
72     /*!
73      * \brief Valeur minimale des chiffres à prédire.
74      *
75      * Cette valeur est incluse dans les chiffres à prédire.
76      */
```

```
77      const unsigned minimum_;
78
79      /*!
80       * \brief Valeur maximale des chiffres à prédire.
81       *
82       * Cette valeur est incluse dans les chiffres à prédire.
83       * Elle doit être au moins égale à \ref minimum_.
84       */
85      const unsigned maximum_;
86
87      /*!
88       * \brief Nombre de chiffres à prédire.
89       *
90       * Cette valeur doit se trouver dans l'intervalle
91       * [ 1, \ref maximum_ - \ref minimum_ + 1 ] :
92       *
93       *    + elle ne peut être nulle ;
94       *    + elle ne peut être supérieure au nombre de chifres
95       *      compris entre les valeurs minimale et maximale.
96       */
97      const unsigned length_;
98
99    public:
100
101     /*!
102      * \brief Constructeur.
103      *
104      * \param length nombre de valeurs à prédire.
105      * \param maximum valeur minimale des valeurs à prédire,
106      *                cette valeur incluse.
107      * \param minimum valeur maximale des valeurs à prédire,
108      *                cette valeur incluse.
109      *
110      * \throw std::invalid_argument si :
111      *          + `maximum` est strictement inférieur à `minimum` ;
112      *          + `lenght` vaut zéro ou dépasse le nombre de
113      *            valeurs comprises entre `minimum` et `maximum`,
114      *            ces valeurs incluses.
115      */
116     constexpr explicit Parameter(unsigned length = LENGTH_DEFAULT_,
117                                  unsigned maximum = MAXIMUM_DEFAULT_,
118                                  unsigned minimum = MINIMUM_DEFAULT_);
119
120     /*!
```

```
121         * \brief Accesseur en lecture du minimum des valeurs à prédire.
122         *
123         * \return minimum des valeurs à prédire.
124         */
125        constexpr unsigned minimum() const;
126
127        /*!
128         * \brief Accesseur en lecture du maximum des valeurs à prédire.
129         *
130         * \return maximum des valeurs à prédire.
131         */
132        constexpr unsigned maximum() const;
133
134        /*!
135         * \brief Accesseur en lecture du nombre de valeurs à prédire.
136         *
137         * \return nombre de valeurs à prédire.
138         */
139        constexpr unsigned length() const;
140
141        /*!
142         * \brief Conversion d'un Parameter en std::string.
143         *
144         * Les valeurs minimale et maximale des valeurs à prédire ainsi
145         * que le nombre de valeurs à prédire sont mis en forme
146         * et convertis en std::string.
147         *
148         * \return représentation du Parameter sous la forme d'une
149         *         std::string.
150         *
151         * \see to_string(const Parameter &)
152         */
153        inline std::string to_string() const;
154    };
155
156    // prototypes
157
158    /*!
159     * \brief Conversion d'un Parameter en std::string.
160     *
161     * Les valeurs minimale et maximale des valeurs à prédire ainsi
162     * que le nombre de valeurs à prédire de `parameter` sont mis
163     * en forme et convertis en std::string.
164     *
```

```
165  * \param parameter Parameter à convertir.
166  *
167  * \return représentation de `parameter` sous la forme d'une
168  *         std::string.
169  *
170  * \see Parameter::to_string()
171  */
172 inline std::string to_string(const Parameter & parameter);
173 
174 /*!
175  * \brief Opérateur d'injection d'un Parameter dans un flux en
176  *        sortie.
177  *
178  * Les valeurs minimale et maximale des valeurs à prédire ainsi
179  * que le nombre de valeurs à prédire de `parameter` sont mis en
180  * forme et injectés dans `out`.
181  *
182  * \param out flux dans lequel injecter `parameter`.
183  * \param parameter Parameter à injecter.
184  *
185  * \return flux après injection.
186  *
187  * \see Parameter::to_string()
188  */
189 inline std::ostream & operator<<(std::ostream & out,
190                                  const Parameter & parameter);
191 
192 // implémentation méthodes inline
193 
194 constexpr Parameter::Parameter(unsigned length, unsigned maximum,
195                                unsigned minimum) :
196     minimum_ { minimum },
197     maximum_ { maximum < minimum_ ?
198                throw std::invalid_argument { "maximum < minimum_" } :
199                maximum },
200     length_ { length == 0 ?
201               throw std::invalid_argument { "length == 0" } :
202               maximum_ - minimum_ + 1 < length ?
203               throw std::invalid_argument { "maximum_ - minimum_ + 1 "
204                                             "< length" } :
205               length}
206 { }
207 
208 constexpr unsigned Parameter::minimum() const
```

```
209 {
210     return minimum_;
211 }
212
213 constexpr unsigned Parameter::maximum() const
214 {
215     return maximum_;
216 }
217
218 constexpr unsigned Parameter::length() const
219 {
220     return length_;
221 }
222
223 std::string Parameter::to_string() const
224 {
225     return std::to_string(length_)
226             .append(" : [")
227             .append(std::to_string(minimum_))
228             .append("..")
229             .append(std::to_string((maximum_)))
230             .append("]");
231 }
232
233 // implémentation fonctions inline
234
235 std::string to_string(const Parameter & parameter)
236 {
237     return parameter.to_string();
238 }
239
240 std::ostream & operator<<(std::ostream & out,
241                           const Parameter & parameter)
242 {
243     return out << parameter.to_string();
244 }
245
246 }   // namespace nvs::lotto
247
248 #endif // PARAMETER_H
```

B. Classe `Item`

```cpp
/*!
 * \file item_incomplete.h
 *
 * \brief Définition de la classe nvs::lotto::Item ainsi que
 *        de fonctions associées.
 */
#ifndef ITEM_INCOMPLETE_H
#define ITEM_INCOMPLETE_H

#include <set>
#include <string>
#include <ostream>
#include <stdexcept>

// TODO : éventuellement ajouter include

#include "parameter.h"

/*!
 * \brief Espace de nom du projet Lotto de Nicolas Vansteenkiste.
 */
namespace nvs::lotto
{

/*!
 * \brief Classe mère des pronostics et tirage d'un jeu de lotto.
 *
 * Il s'agit d'une classe abstraite.
 *
 * Son destructeur est virtuel
 * pur. Cependant, comme ses classes filles auront à utiliser ce
 * destructeur, il doit être implémenté. Vous trouverez davantage
 * d'informations sur les méthodes virtuelles pures _avec_ une
 * implémentation dans le [GotW #31](http://www.gotw.ca/gotw/031.htm).
 * Sa section 2.1 se penche spécifiquement sur le cas des
 * destructeurs virtuels purs.
 */
class Item
{
    /*!
     * \brief Référence vers un paramétrage de grille de lotto.
     */
```

```
43      const Parameter & parameter_;
44
45      /*!
46       * \brief Ensemble de valeurs prédites (dans le cas d'un
47       *        pronostic) ou tirées aléatoirement (dans le cas d'un
48       *        tirage).
49       *
50       * Comme il s'agit d'un std::set, chaque valeur y est unique.
51       */
52      const std::set<unsigned> values_;
53
54      /*!
55       * \brief Vérification de tailles.
56       *
57       * Lors de la création d'un Item, il faut vérifier que le
58       * nombre de valeurs du pronostic, c'est-à-dire le nombre de
59       * valeurs dans \ref values_, correspond bien au paramétrage
60       * du lotto, spécifié via \ref parameter_.
61       *
62       * \throw std::invalid_argument si le nombre de valeurs
63       *          fournies à la construction ne correspond pas au
64       *          nombre de valeurs à prédire pour ce lotto.
65       *
66       * \see Item()
67       */
68      inline void checkLengths() const;
69
70      /*!
71       * \brief Vérification de valeurs.
72       *
73       * Lors de la création d'un Item, il faut vérifier que
74       * les valeurs du pronostic, c'est-à-dire les valeurs
75       * présentes dans \ref values_, correspondent bien au paramétrage
76       * du lotto, spécifié via \ref parameter_.
77       *
78       * \throw std::invalid_argument si les valeurs fournies
79       *          à la construction ne sont pas dans l'intervalle
80       *          des valeurs acceptées pour ce lotto.
81       *
82       * \see Item()
83       */
84      inline void checkValues() const;
85
86  public:
```

```
87
88      /*!
89       * \brief Constructeur générique.
90       *
91       * La référence `parameter` sert à initialiser l'attribut
92       * \ref parameter_,
93       * lui-même référence de Parameter. L'argument doit donc faire
94       * référence à une Parameter qui survit à l'appel du
95       * constructeur.
96       * Dès lors, proscrire un appel comme :
97       *
98       *     Item(values, Parameter { });
99       *
100      * Tout le contenu du conteneur values est utilisé pour
101      * initialiser l'attribut \ref values_. À cette fin, le type
102      * Container doit disposer _dans son espace de nom_ de fonction
103      * nommées `cbegin()` et `cend()`. La première retourne un
104      * itérateur de valeurs constantes sur son premier élément.
105      * La seconde un itérateur passé son dernier élément. Par
106      * ailleurs, comme \ref values_ est un
107      * std::set<unsigned>,  `values` peut ne pas survire à l'appel
108      * du constructeur et son contenu doit pouvoir être
109      * transtypé en `unsigned`. Voici un appel valide :
110      *
111      *     Item(std::vector<double> { 7.7, 2.3, 43 }, parameter);
112      *
113      * Les méthodes checkLengths() et checkValues() sont
114      * appelées pour vérifier que les données fournies via
115      * `values` respectent bien les paramètres de `parameter`.
116      *
117      * \param values conteneur quelconque de valeurs.
118      * \param parameter paramétrage de lotto.
119      *
120      * \throw std::invalid_argument si
121      *          + le nombre de valeurs fournies via `values` ne
122      *            correspond pas au nombre de valeurs à prédire
123      *            selon `parameter` ;
124      *          + les valeurs fournies via `values` ne sont pas
125      *            dans l'intervalle des valeurs acceptables selon
126      *            `parameter`.
127      *
128      * \see checkLengths(), checkValues()
129      */
130     template<typename Container>
```

```
131     inline explicit Item(const Container & values,
132                          const Parameter & parameter);
133
134     /*!
135      * \brief Destructeur virtuel pur.
136      *
137      * Comme la classe est destinée à être héritée, son destructeur
138      * doit être [virtuel]
139      * (https://stackoverflow.com/questions/461203/when-to-use-virtual-destructors,
140      *
141      * L'utilisation de la classe n'a de sens qu'à travers une
142      * classe dérivée. Il s'agit donc d'une [classe abstraite]
143      * (https://en.cppreference.com/w/cpp/language/abstract_class).
144      * Le choix de la méthode virtuelle pure, pour obtenir une
145      * classe abstraite, s'est porté sur le destructeur.
146      *
147      * Cependant, les classes filles vont implicitement utiliser
148      * le destructeur de la classe mère. Le destructeur virtuel
149      * pur requiert donc une implémentation. On trouve plus
150      * d'informations sur les méthodes virtuelles pures dotées
151      * d'une implémentation dans le [GotW #31]
152      * (http://www.gotw.ca/gotw/031.htm). L'implémentation
153      * du [destructeur](https://en.cppreference.com/w/cpp/language/destructor)
154      * par défaut fait ici très bien l'affaire.
155      *
156      * Finalement, il est étiqueté [`inline`]
157      * (https://stackoverflow.com/a/1759575) pour ne pas violer
158      * la [One Definition Rule]
159      * (https://en.wikipedia.org/wiki/One_Definition_Rule).
160      *
161      * \see https://stackoverflow.com/questions/461203/when-to-use-virtual-destruc
162      * \see https://en.cppreference.com/w/cpp/language/abstract_class
163      * \see http://www.gotw.ca/gotw/031.htm
164      * \see https://en.cppreference.com/w/cpp/language/destructor
165      * \see https://stackoverflow.com/a/1759575
166      * \see https://en.wikipedia.org/wiki/One_Definition_Rule
167      */
168     inline virtual ~Item() = 0;
169
170     /*!
171      * \brief Constructeur par recopie par défaut.
172      *
173      * Le destructeur virtuel par défaut a des effets en cascade.
174      *
```

```
175      * \see https://stackoverflow.com/q/33957037
176      * \see https://scottmeyers.blogspot.de/2014/03/a-concern-about-rule-of-zero.ht
177      * \see https://blog.feabhas.com/2015/11/becoming-a-rule-of-zero-hero/
178      */
179     Item(const Item &) = default;
180
181     /*!
182      * \brief Constructeur par déplacement par défaut.
183      *
184      * Le destructeur virtuel par défaut a des effets en cascade.
185      *
186      * \see https://stackoverflow.com/q/33957037
187      * \see https://scottmeyers.blogspot.de/2014/03/a-concern-about-rule-of-zero.ht
188      * \see https://blog.feabhas.com/2015/11/becoming-a-rule-of-zero-hero/
189      */
190     Item(Item &&) = default;
191
192     /*!
193      * \brief Accesseur en lecture des paramètres de la grille de
194      *        lotto.
195      *
196      * \return paramètres de la grille de lotto associée à cet
197      *         élément de lotto.
198      */
199     inline const Parameter & parameter() const;
200
201     /*!
202      * \brief Accesseur en lecture des valeurs.
203      *
204      * Il s'agit des valeurs pronostiquées ou tirées au sort
205      * selon la nature de l'instance.
206      *
207      * \return les valeurs de cet élément de lotto.
208      */
209     inline const std::set<unsigned> & values() const;
210
211     /*!
212      * \brief Conversion d'un Item en std::string.
213      *
214      * Les paramètres de la grille et les valeurs de l'élément
215      * de lotto sont mis en forme et convertis en std::string.
216      *
217      * \return représentation du Item sous la forme d'une
218      *         std::string.
```

```
219         *
220         * \see to_string(const Item &)
221         */
222        inline virtual std::string to_string() const;
223    };
224
225    // prototypes
226
227    /*!
228     * \brief Conversion d'un Item en std::string.
229     *
230     * Les paramètres de la grille et les valeurs de l'élément
231     * de lotto `item` sont mis en forme et convertis en
232     * std::string.
233     *
234     * Pour obtenir un comportement polymorphique sur l'_argument_
235     * `item`, la méthode Item::to_string() est utilisée.
236     *
237     * \param item Item à convertir.
238     *
239     * \return représentation de `item` sous la forme
240     *         d'une std::string.
241     *
242     * \see Item::to_string()
243     */
244    inline std::string to_string(const Item & item);
245
246    /*!
247     * \brief Opérateur d'injection d'un Item dans un flux en
248     *        sortie.
249     *
250     * Les paramètres de la grille et les valeurs de l'élément
251     * de lotto `item` sont mis en forme et injectés dans `out`.
252     *
253     * Pour obtenir un comportement polymorphique sur l'_argument_
254     * `item`, la méthode Item::to_string() est utilisée.
255     *
256     * \param out flux dans lequel injecter `item`.
257     * \param item Item à injecter.
258     *
259     * \return flux après injection.
260     *
261     * \see Item::to_string()
262     */
```

```
263 inline std::ostream & operator<<(std::ostream & out,
264                                   const Item & item);
265
266 // implémentation méthodes inline
267
268 void Item::checkLengths() const
269 {
270     // TODO
271     // throw std::invalid_argument { "size error" };
272 }
273
274 void Item::checkValues() const
275 {
276     // TODO
277     // throw std::invalid_argument { "value error" };
278 }
279
280 template<typename Container>
281 Item::Item(const Container & values, const Parameter & parameter) :
282     parameter_ { parameter },
283     values_ { cbegin(values), cend(values) }
284     // ici cbegin et cend recherchés dans namespace de container
285     // car ADL : https://en.cppreference.com/w/cpp/language/adl
286 {
287     checkLengths();
288     checkValues();
289 }
290
291 // https://stackoverflow.com/a/11494673
292 // http://www.gotw.ca/gotw/031.htm
293 Item::~Item() = default;
294
295 const Parameter & Item::parameter() const
296 {
297     return parameter_;
298 }
299
300 const std::set<unsigned> & Item::values() const
301 {
302     return values_;
303 }
304
305 std::string Item::to_string() const
306 {
```

```
307     std::string result { parameter_.to_string() };
308     result.append(" :: { ");
309     for (auto e : values_)
310     {
311         result.append(std::to_string(e)).append(" ");
312     }
313     return result.append("}");
314 }
315 
316 // implémentation fonctions inline
317 
318 std::string to_string(const Item & item)
319 {
320     return item.to_string();
321 }
322 
323 std::ostream & operator<<(std::ostream & out,
324                           const Item & item)
325 {
326     return out << item.to_string();
327 }
328 
329 }   // namespace nvs::lotto
330 
331 #endif // ITEM_INCOMPLETE_H
```

C. Classe `Draw`

```
1  /*!
2   * \file draw.h
3   *
4   * \brief Définition de la classe nvs::lotto::Draw.
5   */
6  #ifndef DRAW_H
7  #define DRAW_H
8  
9  #include "item.h"
10 #include "parameter.h"
11 
12 #include <set>
13 #include <vector>
14 #include <numeric>
15 #include <algorithm>
```

```cpp
#include <iterator>

#include "random/random.hpp"

/*!
 * \brief Espace de nom du projet Lotto de Nicolas Vansteenkiste.
 */
namespace nvs::lotto
{

/*!
 * \brief Classe représentant un tirage de numéros de lotto.
 */
class Draw : public Item
{
    /*!
     * \brief Méthode réalisant le tirage.
     *
     * Le std::set d'`unsigned` retourné respecte en taille
     * et valeurs les paramètres du jeu de lotto fournis via
     * l'argument `parameter`.
     *
     * \param parameter paramètres du jeu de lotto.
     *
     * \return tirage valide pour le lotto paramétré par `parameter`.
     */
    inline std::set<unsigned> draw(const Parameter & parameter) const;

  public:

    /*!
     * \brief Constructeur d'un tirage au sort de numéros de lotto.
     *
     * La méthode privée draw() fait la boulot dans le respect
     * des paramètres du lotto.
     *
     * \param parameter paramètres du jeu de lotto.
     *
     * \see draw()
     */
    inline explicit Draw(const Parameter & parameter);

    /*!
     * \brief Constructeur virtuel par défaut.
```

```
60      */
61      virtual ~Draw() override = default;
62  };
63
64  // implémentation méthodes inline
65
66  Draw::Draw(const Parameter & parameter) :
67      Item { draw(parameter), parameter }
68      // rem. : on doit fournir parameter à draw car au moment de
69      //        l'appel de draw l'attribut parameter_ n'est
70      //        (peut-être) pas encore construit
71  { }
72
73  std::set<unsigned> Draw::draw(const Parameter & parameter) const
74  {
75      using namespace std;
76      vector<unsigned> universe(parameter.maximum() -
77                                parameter.minimum() + 1);
78      iota(begin(universe), end(universe), parameter.minimum());
79      shuffle(begin(universe), end(universe), nvs::urng());
80      return { cbegin(universe), cbegin(universe) +
81               parameter.length() };
82  }
83
84  }   // namespace nvs::lotto
85
86  #endif // DRAW_H
```

D. Fichier `random.hpp`

```
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 successives
43  * 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 aléatoires
64   * 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.
83   *
84   * Les flottants produits se distribuent uniformément entre
85   * `min` et `max`, la valeur minimale comprise, la maximale non.
86   *
87   * Si `max` est strictement inférieur à `min`, les contenus de ces
88   * variables sont permutés.
89   *
90   * Cette fonction est largement inspirée par Random Number Generation in
91   * C++11 ([WG21 N3551]
92   * (http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2013/n3551.pdf)),
93   * par Walter E. Brown.
94   *
95   * _Remarque_ : Par rapport au modèle de fonction nvs::random_value<T>
96   * produisant des entiers aléatoires, les arguments `min` et `max`
97   * sont inversés de sorte à avoir la valeur nulle (0) comme borne
98   * (minimale ou maximale) si la fonction est appelée avec un seul
99   * argument. Notez que cela n'a pas de réelle incidence sur la
100  * signification des paramètres puisque leurs contenus sont permutés
```

```
 * si nécessaire.
 *
 * \param max la borne supérieure (ou inférieure) de l'intervalle
 *            dans lequel les flottants sont générés.
 * \param min la 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.
 *
 * The effect is undefined if T is not one of : short, int, long,
 * long long, unsigned short, unsigned int, unsigned long, or
 * unsigned long long.
 *
 * 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.
 *
 * \param min la valeur minimale (ou maximale) pouvant être retournée.
 * \param max la valeur maximale (ou minimale) pouvant être retournée.
 *
 * \return un entier entre `min` et `max`.
```

```
 */
template<typename T = int>
inline T random_value(T min = std::numeric_limits<T>::min(),
                      T max = std::numeric_limits<T>::max())
{
    static std::uniform_int_distribution<T> d {};

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

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

} // namespace nvs

#endif // RANDOM_HPP
```

E. Classe `Pronostic`

```
/*!
 * \file pronostic_incomplete.h
 *
 * \brief Définition de la classe nvs::lotto::Pronostic.
 */
#ifndef PRONOSTIC_INCOMPLETE_H
#define PRONOSTIC_INCOMPLETE_H

#include "item.h"
#include "parameter.h"

#include <string>
#include <initializer_list>
#include <set>

/*!
 * \brief Espace de nom du projet Lotto de Nicolas Vansteenkiste.
 */
namespace nvs::lotto
{

/*!
 * \brief Classe représentant un pronostic de jeu de lotto.
 */
```

```cpp
class Pronostic : public Item
{
    /*!
     * \brief Nom / identifiant de l'auteur / propriétaire
     *        du pronostic.
     */
    const std::string owner_;

  public:

    /*!
     * \brief Constructeur générique.
     *
     * À part pour ce qui concerne le propriétaire du pronostic,
     * qui peut être transmis à l'aide d'une valeur temporaire
     * car sa valeur est clonée dans l'attribut \ref owner_,
     * tout a été dit dans la description du constructeur de
     * Item Item::Item().
     *
     * Un appel correct est :
     *
     *     Pronostic { "owner", std::vector<unsigned> { 4, 6, 15, 12 }, parameter }
     *
     * \param owner nom / identifiant du propriétaire du pronostic.
     * \param values conteneur quelconque de valeurs.
     * \param parameter paramétrage de lotto.
     *
     * \throw std::invalid_argument si
     *          + le nombre de valeurs fournies via `values` ne
     *            correspond pas au nombre de valeurs à prédire
     *            selon `parameter` ;
     *          + les valeurs fournies via `values` ne sont pas
     *            dans l'intervalle des valeurs acceptables selon
     *            `parameter`.
     *
     * \see Item::Item()
     */
    template<typename Container>
    explicit inline Pronostic(const std::string & owner,
                              const Container & values,
                              const Parameter & parameter);

    /*!
     * \brief Constructeur générique avec une
```

```
69      *        liste d'initialisation en argument.
70      *
71      * Il s'agit d'une alternative pour permettre la création
72      * d'un pronostic avec une liste d'initialisation pour
73      * renseigner les valeurs.
74      *
75      * Voici un appel correct :
76      *
77      *     Pronostic { "owner", { 4, 6, 15, 12 }, parameter }
78      *
79      * \param owner nom / identifiant du propriétaire du pronostic.
80      * \param values liste d'initialisation avec les valeurs.
81      * \param parameter paramétrage de lotto.
82      *
83      * \throw std::invalid_argument si
84      *          + le nombre de valeurs fournies via `values` ne
85      *            correspond pas au nombre de valeurs à prédire
86      *            selon `parameter` ;
87      *          + les valeurs fournies via `values` ne sont pas
88      *            dans l'intervalle des valeurs acceptables selon
89      *            `parameter`.
90      *
91      * \see Pronostic::Pronostic(const std::string &, const Container &, const Para
92      */
93     template<typename T>
94     explicit inline Pronostic(const std::string & owner,
95                               const std::initializer_list<T> & values,
96                               const Parameter & parameter);
97
98     /*!
99      * \brief Constructeur générique avec un
100     *        vieux tableau du C en argument.
101     *
102     * Il s'agit d'une alternative pour permettre la création
103     * d'un pronostic avec un vieux tableau pour
104     * renseigner les valeurs.
105     *
106     * Voici un appel correct :
107     *
108     *     int t [] = { 4, 6, 15, 12 };
109     *     Pronostic { "owner", t, parameter }
110     *
111     * \param owner nom / identifiant du propriétaire du pronostic.
112     * \param values vieux tableau avec les valeurs.
```

```
113         * \param parameter paramétrage de lotto.
114         *
115         * \throw std::invalid_argument si
116         *          + le nombre de valeurs fournies via `values` ne
117         *            correspond pas au nombre de valeurs à prédire
118         *            selon `parameter` ;
119         *          + les valeurs fournies via `values` ne sont pas
120         *            dans l'intervalle des valeurs acceptables selon
121         *            `parameter`.
122         *
123         * \see Pronostic::Pronostic(const std::string &, const Container &, const Para
124         */
125        template<typename T, std::size_t N>
126        explicit inline Pronostic(const std::string & owner,
127                                  const T (& values) [N],
128                                  const Parameter & parameter);
129
130        /*!
131         * \brief Constructeur virtuel par défaut.
132         */
133        virtual ~Pronostic() override = default;
134
135        /*!
136         * \brief Accesseur en lecture du propriétaire du pronostic.
137         *
138         * \return propriétaire du pronostic.
139         */
140        inline const std::string & owner() const;
141
142        /*!
143         * \brief Conversion d'un Pronostic en std::string.
144         *
145         * Le propriétaire, les paramètres de la grille et les valeurs
146         * du pronostic sont mis en forme et convertis en std::string.
147         *
148         * \return représentation du Pronostic sous la forme d'une
149         *         std::string.
150         *
151         * \see to_string(const Pronostic &)
152         */
153        inline std::string to_string() const override;
154    };
155
156    // implémentation méthodes inline
```

```
157
158 template<typename Container>
159 Pronostic::Pronostic(const std::string & owner,
160                      const Container & values,
161                      const Parameter & parameter) :
162     // TODO
163 { }
164
165 template<typename T>
166 Pronostic::Pronostic(const std::string & owner,
167                      const std::initializer_list<T> & values,
168                      const Parameter & parameter) :
169     Pronostic { owner,
170                 std::set<unsigned> { cbegin(values),
171                     cend(values) }, parameter }
172 { }
173
174 template<typename T, std::size_t N>
175 Pronostic::Pronostic(const std::string & owner,
176                      const T (& values) [N],
177                      const Parameter & parameter) :
178     Pronostic { owner,
179                 std::set<unsigned> { std::cbegin(values),
180                     std::cend(values) }, parameter }
181 { }
182
183 const std::string & Pronostic::owner() const
184 {
185     return owner_;
186 }
187
188 std::string Pronostic::to_string() const
189 {
190         return Item::to_string().append(" ::: ").append(owner_);
191 }
192
193 }   // namespace nvs::lotto
194
195 #endif // PRONOSTIC_INCOMPLETE_H
```

F. Classe `Lotto`

```cpp
/*!
 * \file lotto_incomplete.h
 *
 * \brief Définition de la classe nvs::lotto::Lotto ainsi que
 *        de fonctions associées.
 */
#ifndef LOTTO_INCOMPLETE_H
#define LOTTO_INCOMPLETE_H

#include "parameter.h"
#include "pronostic.h"
#include "draw.h"

#include <vector>
#include <optional>
#include <string>
#include <ostream>

// TODO : ajouter d'éventuels include


/*!
 * \brief Espace de nom du projet Lotto de Nicolas Vansteenkiste.
 */
namespace nvs::lotto
{

/*!
 * \brief Classe gérant un jeu de lotto.
 *
 * À sa création, on spécifie les valeurs minimale et maximale
 * de la grille et le nombre de valeurs d'un tirage / d'un
 * pronostic.
 *
 * Ensuite on enregistre les pronostics.
 *
 * Puis le tirage au sort des la combinaison gagnante est
 * réalisé.
 *
 * Enfin, on peut déterminer les gagants.
 */
class Lotto
```

```
43  {
44      /*!
45       * \brief Paramétrage du lotto : valeurs minimale et maximale
46       *        et nombre de valeurs d'un tirage / pronostic.
47       */
48      const Parameter parameter_;
49
50      /*!
51       * \brief Ensemble des pronostics de ce jeu de lotto.
52       */
53      std::vector<Pronostic> pronostics_;
54
55      /*!
56       * \brief Tirage de ce jeu de lotto.
57       */
58      std::optional<Draw> draw_;
59
60    public:
61
62      /*!
63       * \brief Constructeur.
64       *
65       * Le paramétrage de lotto est construit. Davantage de
66       * détail dans le constructeur de Parameter
67       * Parameter::Parameter().
68       *
69       * Le lotto est construit sans pronostic et sans tirage.
70       *
71       * \param length nombre de valeurs à prédire.
72       * \param maximum valeur minimale des valeurs à prédire,
73       *                cette valeur incluse.
74       * \param minimum valeur maximale des valeurs à prédire,
75       *                cette valeur incluse.
76       *
77       * \throw std::invalid_argument si :
78       *          + `maximum` est strictement inférieur à `minimum` ;
79       *          + `lenght` vaut zéro ou dépasse le nombre de
80       *            valeurs comprises entre `minimum` et `maximum`,
81       *            ces valeurs incluses.
82       *
83       * \see Parameter::Parameter()
84       */
85      inline explicit Lotto(unsigned length = Parameter::LENGTH_DEFAULT_,
86                            unsigned maximum = Parameter::MAXIMUM_DEFAULT_,
```

```
87                          unsigned minimum = Parameter::MINIMUM_DEFAULT_);
88
89      /*!
90       * \brief Accesseur en lecture des paramètres de la grille de
91       *        lotto.
92       *
93       * \return paramètres de lotto associée à ce jeu de lotto.
94       */
95      inline const Parameter & parameter() const;
96
97      /*!
98       * \brief Accesseur en lecture de l'ensemble des pronostics
99       *        réalisés.
100      *
101      * \return ensemble des pronostics.
102      */
103     inline const std::vector<Pronostic> & pronostics() const;
104
105     /*!
106      * \brief Accesseur en écriture d'un pronostic.
107      *
108      * Le pronostic est associé aux paramètres du jeu de lotto. Il
109      * est construit à l'aide du constructeur
110      * Pronostic::Pronostic(const std::string &, const Container &, const Parameter
111      *
112      * Il est interdit d'ajouter un pronostic _après_ le tirage.
113      *
114      * \param owner propriétaire du pronostic.
115      * \param values valeurs du pronostic.
116      *
117      * \return le jeu de lotto.
118      *
119      * \throw std::logic_error si le tirage a déjà été réalisé.
120      *
121      * \throw std::invalid_argument si
122      *          + le nombre de valeurs fournies via `values` ne
123      *            correspond pas au nombre de valeurs à prédire
124      *            selon les paramètres de la grille du lotto ;
125      *          + les valeurs fournies via `values` ne sont pas
126      *            dans l'intervalle des valeurs acceptables selon
127      *            les paramètres de la grille du lotto.
128      *
129      * \see Pronostic::Pronostic(const std::string &, const Container &, const Para
130      */
```

```
131     template<typename Container>
132     inline Lotto & add(std::string owner, const Container & values);
133
134     // rnvs : alternative avec initializer_list nécessaire ou pas ?
135
136     /*!
137      * \brief Accesseur en écriture d'un pronostic.
138      *
139      * Alternative à la méthode add(std::string, const Container &)
140      * avec un vieux tableau du C (ou une liste d'initialisation)
141      * pour fournir les valeurs.
142      *
143      * Le pronostic est associé aux paramètres de la grille du jeu
144      * de lotto. Il
145      * est construit à l'aide du constructeur
146      * Pronostic::Pronostic(const std::string &, const T (&) [N], const Parameter &
147      *
148      * Il est interdit d'ajouter un pronostic _après_ le tirage.
149      *
150      * \param owner propriétaire du pronostic.
151      * \param values valeurs du pronostic.
152      *
153      * \return le jeu de lotto.
154      *
155      * \throw std::logic_error si le tirage a déjà été réalisé.
156      *
157      * \throw std::invalid_argument si
158      *          + le nombre de valeurs fournies via `values` ne
159      *            correspond pas au nombre de valeurs à prédire
160      *            selon les paramètres de la grille du lotto ;
161      *          + les valeurs fournies via `values` ne sont pas
162      *            dans l'intervalle des valeurs acceptables selon
163      *            les paramètres de la grille du lotto.
164      *
165      * \see Pronostic::Pronostic(const std::string &, const T (&) [N], const Parame
166      */
167     template<typename T, std::size_t N>
168     inline Lotto & add(std::string owner, const T (& values) [N]);
169
170     /*!
171      * \brief Accesseur en écriture d'un pronostic.
172      *
173      * Le pronostic doit être associé au même paramétrage que celui
174      * du jeu de lotto.
```

```
175         *
176         * Il est interdit d'ajouter un pronostic _après_ le tirage.
177         *
178         * \param pronostic pronostic à ajouter à l'ensemble des
179         *        pronostics.
180         *
181         * \return le jeu de lotto.
182         *
183         * \throw std::logic_error si le tirage a déjà été réalisé.
184         *
185         * \throw std::invalid_argument si le paramétrage de la grille
186         *                              du pronostic n'est
187         *                              pas celui du lotto.
188         */
189        inline Lotto & add(const Pronostic & pronostic);
190
191        /*!
192         * \brief Opérateur pour l'accès en écriture d'un pronostic.
193         *
194         * Le pronostic doit être associé au même paramétrage que celui
195         * du jeu de lotto.
196         *
197         * Il est interdit d'ajouter un pronostic _après_ le tirage.
198         *
199         * \param pronostic pronostic à ajouter à l'ensemble des
200         *        pronostics.
201         *
202         * \return le jeu de lotto.
203         *
204         * \throw std::logic_error si le tirage a déjà été réalisé.
205         *
206         * \throw std::invalid_argument si le paramétrage de la grille
207         *                              du pronostic n'est
208         *                              pas celui du lotto.
209         *
210         * \see add(const Pronostic &)
211         */
212        inline Lotto & operator+=(const Pronostic & pronostic);
213
214        /*!
215         * \brief Accesseur en lecture pour savoir si le tirage a eu
216         *        lieu.
217         *
218         * \return `false` si le tirage n'a pas eu lieu, `true` à
```

```
219         *          partir du moment où il a eu lieu.
220         */
221        inline bool has_draw() const;
222
223        /*!
224         * \brief Accesseur en écriture pour réaliser le tirage.
225         *
226         * Le tirage correspond évidemment aux paramètres du lotto.
227         * Il ne peut y avoir qu'un seul tirage.
228         *
229         * \throw std::logic_error si le tirage a déjà eu lieu.
230         */
231        inline void set_draw();
232
233        /*!
234         * \brief Accesseur en lecture du tirage.
235         *
236         * \return un clone du tirage, s'il a eu lieu.
237         *
238         * \throw std::logic_error si le tirage n'a pas déjà été
239         *        réalisé.
240         */
241        inline Draw draw() const;
242
243        /*!
244         * \brief Énumération fortement typée pour indiquer le type
245         *        de l'égalité à utiliser par la méthode
246         *        winner().
247         */
248        enum class Equality
249        {
250            /*! Pour indiquer l'égalité stricte. */
251            STRICT,
252            /*! Pour indiquer l'égalité inclusive. */
253            INCLUSIVE
254        };
255
256        /*!
257         * \brief Accesseur en lecture des pronostics gagnants.
258         *
259         * Cette méthode ne peut être invoquée qu'après le
260         * tirage réalisé.
261         *
262         * Elle retourne l'ensemble des pronostics
```

```
263       * comportant :
264       *
265       *  + _exactement_ `predicted_length` chiffres présents
266       *    dans le tirage lorsque la valeur de `type` est
267       *    Equality::STRICT ;
268       *  + _au moins_ `predicted_length` chiffres présents dans
269       *    le tirage lorsque la valeur de `type` est
270       *    Equality::INCLUSIVE.
271       *
272       * \param predicted_length le nombre de chiffres du tirage
273       *                         présents dans le pronostic.
274       * \param type le type d'égalité utilisée (voir ci-dessus).
275       *
276       * \return ensemble des pronostics répondant aux paramètres.
277       *
278       * \throw std::logic_error si le tirage n'a pas encore eu lieu.
279       *
280       * \throw std::invalid_argument si `predicted_length` est nul
281       *          ou strictement supérieur au nombre de valeurs du
282       *          tirage.
283       */
284      inline std::vector<Pronostic>
285      winner(unsigned predicted_length,
286             Equality type = Equality::STRICT) const;
287
288      /*!
289       * \brief Conversion d'un Lotto en std::string.
290       *
291       * Les paramètres de la grille (minimum, maximum et taille)
292       * sont toujours convertis en std::string.
293       *
294       * Si le tirage a eu lieu, son résultat et l'ensemble des
295       * pronostics sont convertis. Sinon, un message indiquant
296       * que le tirage n'a pas encore eu lieu est ajouté à la
297       * std::string.
298       *
299       * \return représentation du Lotto sous la forme d'une
300       *         std::string.
301       *
302       * \see to_string(const Lotto &)
303       */
304      inline std::string to_string() const;
305  };
306
```

```
307 // prototypes
308
309 /*!
310  * \brief Conversion d'un Lotto en std::string.
311  *
312  * Les paramètres de la grille (minimum, maximum et taille) de
313  * `lotto` sont toujours convertis en std::string.
314  *
315  * Si le tirage de `lotto` a eu lieu, son résultat et l'ensemble de
316  * ses pronostics sont convertis. Sinon, un message indiquant
317  * que le tirage n'a pas encore eu lieu est ajouté à la
318  * std::string.
319  *
320  * \param lotto Lotto à convertir.
321  *
322  * \return représentation de `lotto` sous la forme d'une
323  *         std::string.
324  *
325  * \see Lotto::to_string()
326  */
327 inline std::string to_string(const Lotto & lotto);
328
329 /*!
330  * \brief Opérateur d'injection d'un Lotto dans un flux en sortie.
331  *
332  * Les paramètres de la grille (minimum, maximum et taille) de
333  * `lotto` sont toujours injectés dans `out`.
334  *
335  * Si le tirage de `lotto` a eu lieu, son résultat et l'ensemble de
336  * ses pronostics sont injectés. Sinon, un message indiquant
337  * que le tirage n'a pas encore eu lieu est injectés dans `out`.
338  *
339  * \param out flux dans lequel injecter `lotto`.
340  * \param lotto Lotto à injecter.
341  *
342  * \return flux après injection.
343  *
344  * \see Lotto::to_string()
345  */
346 inline std::ostream & operator<<(std::ostream & out,
347                                  const Lotto & lotto);
348
349 // implémentation méthodes inline
350
```

```
351 Lotto::Lotto(unsigned length, unsigned maximum, unsigned minimum) :
352     // TODO
353 { }
354
355 const Parameter & Lotto::parameter() const
356 {
357     return parameter_;
358 }
359
360 const std::vector<Pronostic> & Lotto::pronostics() const
361 {
362     return pronostics_;
363 }
364
365 template<typename Container>
366 inline Lotto & Lotto::add(std::string owner,
367                           const Container & values)
368 {
369     return add(Pronostic { owner, values, parameter_ });
370 }
371
372 template<typename T, std::size_t N>
373 Lotto & Lotto::add(std::string owner, const T (& values) [N])
374 {
375     return add(Pronostic { owner, values, parameter_ });
376 }
377
378 Lotto & Lotto::add(const Pronostic & pronostic)
379 {
380     // TODO
381 }
382
383 Lotto & Lotto::operator+=(const Pronostic & pronostic)
384 {
385     return add(pronostic);
386 }
387
388 bool Lotto::has_draw() const
389 {
390     // TODO
391 }
392
393 void Lotto::set_draw()
394 {
```

```
395     // TODO
396 }
397
398 Draw Lotto::draw() const
399 {
400     // TODO
401 }
402
403 std::vector<Pronostic>
404 Lotto::winner(unsigned predicted_length, Equality type) const
405 {
406     // TODO
407 }
408
409 std::string Lotto::to_string() const
410 {
411     std::string result { "{ g } " };
412     result
413     .append(parameter_.to_string())
414     .append("\n");
415
416     if (has_draw())
417     {
418         // ici tirage a eu lieu
419         result
420         .append("{ d } ")
421         .append(draw_->to_string())
422         .append("\n");
423
424         result
425         .append("{ p } ")
426         .append(std::to_string(pronostics_.size()))
427         .append("\n");
428         for (const auto & e : pronostics_)
429         {
430             result
431             .append("      ")
432             .append(e.to_string())
433             .append("\n");
434         }
435     }
436     else
437     {
438         // ici pas encore de tirage
```

```
439         result.append("no draw yet");
440     }
441
442     return result;
443 }
444
445 // implémentation fonctions inline
446
447 std::string to_string(const Lotto & lotto)
448 {
449     return lotto.to_string();
450 }
451
452 std::ostream & operator<<(std::ostream & out, const Lotto & lotto)
453 {
454     return out << lotto.to_string();
455 }
456
457 }   // namespace nvs::lotto
458
459 #endif // LOTTO_INCOMPLETE_H
```

G. Fonctions de génération de données

G.1. Fichier d'en-têtes

```
1  /*!
2   * \file data.h
3   * \brief Définition d'une fonction pour la production de
4   *        pronostics de lotto.
5   */
6  #ifndef DATA_H
7  #define DATA_H
8
9  #include <vector>
10 #include <utility>
11 #include <string>
12
13 /*!
14  * \brief Espace de nom de Nicolas Vansteenkiste.
15  */
16 namespace nvs
17 {
```

```
18
19 /*!
20  * \brief Espace de nom du projet _Lotto_.
21  */
22 namespace lotto
23 {
24
25 /*!
26  * \brief Fonction qui retourne des valeurs pour produire des
27  *        pronostics pour un jeu de lotto.
28  *
29  * Dans le retour :
30  *
31  *    + la partie std::string peut servir à représenter le
32  *      propriétaire d'un pronostic ;
33  *    + la partie std::vector<unsigned> peut servir à représenter
34  *      les valeurs d'un pronostic.
35  *
36  * \param pronostic_count nombre de pronostics désirés.
37  * \param grid_length nombre de valeurs dans un pronostic.
38  * \param grid_maximum valeur maximale possible des valeurs du
39  *                     pronostic.
40  * \param grid_minimum valeur minimale possible des valeurs du
41  *                     pronostic.
42  *
43  * \return des données pour produire les parties «propriétaire »
44  *         et « valeurs » d'un pronostics de lotto de paramétrage
45  *         donné en argument.
46  *
47  * \throw std::invalid_argument dans les trois cas suivants :
48  *
49  *  + le nombre de valeurs d'un pronostic est nul ;
50  *  + la valeur maximale possible d'un pronostic est strictement
51  *    inférieure à sa valeur minimale possible ;
52  *  + le nombre de valeurs d'un pronostic dépasse le nombre de
53  *    valeurs de l'intervalle des valeurs acceptables.
54  */
55 std::vector<std::pair<std::string, std::vector<unsigned>>>
56 data(unsigned pronostic_count, unsigned grid_length,
57      unsigned grid_maximum, unsigned grid_minimum);
58
59 } // namespace lotto
60
61 } // namespace nvs
```

```
62
63 #endif // DATA_H
```

G.2. Fichier source

```
1  #include <vector>
2  #include <utility>
3  #include <string>
4  #include <stdexcept>
5  #include <numeric>
6  #include <algorithm>
7  #include <iterator>
8
9  #include "../random/random.hpp"
10
11 namespace nvs
12 {
13
14 namespace lotto
15 {
16
17 std::vector<std::pair<std::string, std::vector<unsigned>>>
18 data(unsigned pronostic_count, unsigned grid_length,
19      unsigned grid_maximum, unsigned grid_minimum)
20 {
21     if (grid_length == 0 ||
22             grid_maximum < grid_minimum ||
23             grid_maximum - grid_minimum + 1 < grid_length)
24     {
25         throw std::invalid_argument { "boom !" };
26     }
27
28     static const std::vector<std::string> owner
29     {
30         "38708", "39238", "39864", "41298", "41326", "41949", "42176",
31         "42258", "42265", "42392", "42394", "42482", "42835", "42933",
32         "42945", "42969", "42971", "42991", "43004", "43009", "43015",
33         "43093", "43121", "43256", "43268", "43272", "43308", "43338",
34         "43353", "43370", "43382", "43398", "43453", "43466", "43612",
35         "44343", "44422", "44423", "44424", "44582", "44902", "44942",
36         "45022", "45242", "45324", "45582", "45682", "47802", "47923",
37         "47942", "48022", "48442", "48982", "49102", "49262", "49282",
38         "49382", "49702", "49709", "49720", "49732", "49736", "49737",
```

```
        "49738", "49739", "49762", "49773", "49778", "49791", "49845",
        "49853", "49869", "49877", "49921", "49923", "51395", "51426",
        "51531", "51594", "51603", "51885", "53785",
    };
    static const unsigned owner_last_index
    { static_cast<unsigned>(owner.size()) - 1 };

    std::vector<unsigned> pronostic(grid_maximum - grid_minimum + 1);
    std::iota(std::begin(pronostic), std::end(pronostic),
              grid_minimum);

    decltype (data(0, 0, 0, 0)) result(pronostic_count);

    for (std::size_t i { 0 }; i < pronostic_count; ++i)
    {
        result[i].first = owner[nvs::random_value(0u, owner_last_index)];

        std::shuffle(std::begin(pronostic), std::end(pronostic),
                     nvs::urng());
        result[i].second = { std::cbegin(pronostic),
                             std::cbegin(pronostic) + grid_length
                           };
    }

    return result;
}

} // namespace lotto

} // namespace nvs
```