```
 1: /*
 2:  * BoundedRandomGenerator.h
 3:  *
 4:  *  Created on: 21 fÃ©vr. 2013
 5:  *      Author: davidroussel
 6:  */
 7:
 8: #ifndef BOUNDEDGENERATOR_H_
 9: #define BOUNDEDGENERATOR_H_
10:
11: /**
12:  * Foncteur permettant de gÃ©nÃ©rer des valeurs alÃ©atoires comprises entre un
13:  * minimum et un maximum
14:  * @tparam le type des valeurs
15:  */
16: template <class T>
17: class BoundedRandomGenerator
18: {
19:     private:
20:         /*
21:          * TODO attributs : min & max
22:          */
23:     public:
24:         /*
25:          * TODO mÃ©thodes
26:          *  - constructeur valuÃ© (min, max)
27:          *  - operator ()() : T
28:          */
29: };
30:
31: #endif /* BOUNDEDGENERATOR_H_ */
```



```
 1: /*
 2:  * BoundedRandomGenerator.cpp
 3:  *
 4:  *  Created on: 21 fÃ©vr. 2013
 5:  *      Author: davidroussel
 6:  */
 7:
 8: #include "BoundedRandomGenerator.h"
 9: #include <cstdlib>  // for random & srandom
10: #include <ctime>    // for time
11:
12: /*
13:  * TODO implÃ©mentation des mÃ©thodes
14:  * template <class T>
15:  * retour BoundedRandomGenerator<T>::mÃ©thode(args...)
16:  * {
17:  *  ...
18:  * }
19:  * N'oubliez pas les const Ã  chaque fois que nÃ©cessaire dans
20:  *  - les arguments : mamÃ©thode(const Type & v)
21:  *  - les types de retour : const Type mamÃ©thode(...)
22:  *  - le qualificateur de la mÃ©thode : mamÃ©thode(...) const
23:  */
24:
25: // ----------------------------------------------------------------------------
26: // Proto instanciations
27: // ----------------------------------------------------------------------------
28: /*
29:  * Instanciation du template BoundedRandomGenerator avec des <int>
30:  */
31: template class BoundedRandomGenerator<int>;
32:
33: /*
34:  * Instanciation du template BoundedRandomGenerator avec des <float>
35:  */
36: template class BoundedRandomGenerator<float>;
37:
38: /*
39:  * Instanciation du template BoundedRandomGenerator avec des <double>
40:  */
41: template class BoundedRandomGenerator<double>;
```

```cpp
/*
 * StatFunctor.h
 *
 *  Created on: 21 fÃ©vr. 2013
 *      Author: davidroussel
 */

#ifndef STATFUNCTOR_H_
#define STATFUNCTOR_H_

#include <functional>   // for unary_function
using namespace std;

#include "TypeException.h"

/**
 * Foncteur unaire pour calculer la moyenne et l'Ã©cart type d'un ensemble
 * de valeurs
 * @tparam T type des valeurs
 * @note Attention, la valeur moyenne et l'Ã©cart type doivent renvoyer des
 * valeurs "double" quel que soit le type T.
 */
template<class T>
class StatFunctor: public unary_function<T, void>
{
    private:
        /*
         * TODO attributs
         */
    public:
        /*
         * TODO mÃ©thodes
         *  - constructeur --> throws TypeException si T n'est pas is_arithmetic<T>
         *  - operator ()(const T&) : void
         *  - mean() : double
         *  - std() : double
         *  - reset()
         */
};

#endif /* STATFUNCTOR_H_ */
```



```cpp
/*
 * StatFunctor.cpp
 *
 *  Created on: 21 fÃ©vr. 2013
 *      Author: davidroussel
 */

#include "StatFunctor.h"

#include <cmath>    // for sqrt

/*
 * TODO implÃ©mentation des mÃ©thodes
 * template <class T>
 * retour StatFunctor<T>::mÃ©thode(args...)
 * {
 *  ...
 * }
 * N'oubliez pas les const Ã  chaque fois que nÃ©cessaire dans
 *  - les arguments
 *  - les types de retour
 *  - le qualificateur de la mÃ©thode
 */

// ----------------------------------------------------------------------------
// Proto instanciations
// ----------------------------------------------------------------------------
/*
 * Instanciation du template StatFunctor avec des <int>
 */
template class StatFunctor<int>;

/*
 * Instanciation du template StatFunctor avec des <float>
 */
template class StatFunctor<float>;

/*
 * Instanciation du template StatFunctor avec des <double>
 */
template class StatFunctor<double>;
```

```
 1: /*
 2:  * ComplexException.h
 3:  *
 4:  *  Created on: Nov 2, 2009
 5:  *      Author: david.roussel
 6:  */
 7:
 8: #ifndef EXCEPTION_H_
 9: #define EXCEPTION_H_
10:
11: #include <iostream>     // pour les flux d'entrÃ©e/sortie
12: #include <string>       // pour les string
13: #include <exception>    // pour la classe de base des exceptions
14: using namespace std;
15:
16: /**
17:  * Classe d'exception Ã  utiliser lorsqu'un type T n'est pas compatible avec
18:  * les opÃ©rations que l'on prÃ©voit de faire sur ce type T
19:  */
20: class TypeException : public exception
21: {
22:     protected:
23:         /**
24:          * Le message de l'exception : la raison de sa crÃ©ation
25:          */
26:         string message;
27:     public:
28:         /**
29:          * Constructeur de l'exception
30:          * @param message le message de l'exception
31:          */
32:         TypeException(string &&  message);
33:
34:         /**
35:          * Destructeur de l'exception : efface le message
36:          */
37:         virtual ~TypeException() throw ();
38:
39:         /**
40:          * Description de l'exception
41:          * @return une chaine de caractÃ¨res dÃ©crivant l'exception
42:          */
43:         virtual const char* what() const throw ();
44:
45:         /**
46:          * OpÃ©rateur de sortie standard pour ce type d'exception
47:          * @param out le flux de sortie
48:          * @param c l'exception Ã  afficher
49:          * @return une rÃ©fÃ©rence vers le flux de sortie
50:          */
51:         friend ostream & operator << (ostream & out, const TypeException & c);
52: };
53:
54: #endif /* EXCEPTION_H_ */
```



```
 1: /*
 2:  * ComplexException.cpp
 3:  *
 4:  *  Created on: Nov 2, 2009
 5:  *      Author: david.roussel
 6:  */
 7:
 8: #include "TypeException.h"
 9:
10: /*
11:  * Constructeur de l'exception
12:  * @param message le message de l'exception
13:  */
14: TypeException::TypeException(string && message) :
15:     message(std::forward<string>(message))
16: {
17: }
18:
19: /*
20:  * Destructeur de l'exception : efface le message
21:  */
22: TypeException::~TypeException() throw ()
23: {
24:     message.clear();
25: }
26:
27: /*
28:  * Description de l'exception
29:  * @return une chaine de caractÃ¨res dÃ©crivant l'exception
30:  */
31: const char* TypeException::what() const throw()
32: {
33:     return message.c_str();
34: }
35:
36: /*
37:  * OpÃ©rateur de sortie standard pour ce type d'exception
38:  * @param out le flux de sortie
39:  * @param c l'exception Ã  afficher
40:  * @return une rÃ©fÃ©rence vers le flux de sortie
41:  */
42: ostream & operator << (ostream & out, const TypeException & c)
43: {
44:     out << "TypeException::" << c.message;
45:     return out;
46: }
```

```
/*
 * ExternalCheck.h
 *
 *  Created on: 9 fÃ©vr. 2017
 *      Author: davidroussel
 */

#ifndef EXTERNALCHECK_H_
#define EXTERNALCHECK_H_

#include <string>

using namespace std;

/**
 * Check values provided for mean, std and median using external math tools
 * such as matlab or R
 */
template<class Iterator, class T>
class ExternalCheck
{
    protected:
        /**
         * Internal Iterator's value type
         */
        using value_t = typename iterator_traits<Iterator>::value_type;
        using diff_t = typename iterator_traits<Iterator>::difference_type;

        /**
         *  The tool to launch for this checker
         */
        string tool;

        /**
         * Check the availability to the required tool
         */
        bool availability;

        /**
         * The command string to be sent to the external tool
         */
        string command;

        /**
         * Protected constructor for external checker
         * @param tool the tool to launch for this checker
         * @param baseDir the base directory to run the external tool from
         * @param begin an iterator to the first value of the tested sequence
         * @param end an iterator past the last value of the tested sequence
         * @param minValue minimum possible value in the sequence
         * @param maxValue maximum possible value in the sequence
         * @param meanValue computed mean value of the sequence
         * @param stdValue computed std value of the sequence
         * @param medianValue computed median value in the sequence
         */
        ExternalCheck(const string & tool,
                      const string & baseDir,
                      const Iterator & begin,
                      const Iterator & end,
                      const T & minValue,
                      const T & maxValue,
                      const double & meanValue,
                      const double & stdValue,
                      const double & medianValue);
    public:
        /**
         * Destructor
         */
        virtual ~ExternalCheck();

        /**
         * Check if the required tool is available
         * @return true if the required tool has been found in the path,
         * false otherwise.
         */
        bool available();

        /**
         * Executes the command in a new shell and return value
         * @return the return value of the system call with "command" argument
```



```
         * @pre The tool is supposed to be available before launching the run
         */
        int run();
};

#endif /* EXTERNALCHECK_H_ */
```

```
  1: /*
  2:  * ExternalCheck.cpp
  3:  *
  4:  *  Created on: 9 fÃ©vr. 2017
  5:  *      Author: davidroussel
  6:  */
  7:
  8: //#include <cstdlib>    // for system command
  9: #include <cstdio>   // for popen
 10: #include <iostream> // for cerr
 11: #include <sstream>  // for ostringstream
 12: #include <array>    // for array
 13: #include <memory>   // for shared pointer
 14:
 15: #include "ExternalCheck.h"
 16:
 17: /*
 18:  * Protected constructor for external checker
 19:  * @param tool the tool to launch for this checker
 20:  * @param baseDir the base directory to run the external tool from
 21:  * @param begin an iterator to the first value of the tested sequence
 22:  * @param end an iterator past the last value of the tested sequence
 23:  * @param minValue minimum possible value in the sequence
 24:  * @param maxValue maximum possible value in the sequence
 25:  * @param meanValue computed mean value of the sequence
 26:  * @param stdValue computed std value of the sequence
 27:  * @param medianValue computed median value in the sequence
 28:  */
 29: template<class Iterator, class T>
 30: ExternalCheck<Iterator, T>::ExternalCheck(const string & tool,
 31:                                           const string &,
 32:                                           const Iterator &,
 33:                                           const Iterator &,
 34:                                           const T &,
 35:                                           const T &,
 36:                                           const double &,
 37:                                           const double &,
 38:                                           const double &) :
 39:     tool(tool),
 40:     availability(false)
 41: {
 42:     // Check if this tool is available on the path
 43:     ostringstream oss;
 44:     oss << "command -v " << tool;
 45:
 46:     const int bufSize = 128;
 47:     char buffer[bufSize];
 48:     ostringstream resultStream;
 49:     const char * testcommand = oss.str().c_str();
 50:
 51:     FILE * pipe = popen(testcommand, "r");
 52:     if (pipe)
 53:     {
 54:         while(fgets(buffer, bufSize, pipe) != nullptr)
 55:         {
 56:             resultStream << buffer;
 57:         }
 58:
 59:         if (!resultStream.str().empty())
 60:         {
 61:             availability = true;
 62:         }
 63:         else
 64:         {
 65:             cerr << "command " << tool << " is not available" << endl;
 66:         }
 67:
 68:         pclose(pipe);
 69:     }
 70:     else
 71:     {
 72:         cerr << "popen(" << testcommand << ") failed" << endl;
 73:     }
 74: }
 75:
 76: /*
 77:  * Destructor
 78:  */
 79: template<class Iterator, class T>
 80: ExternalCheck<Iterator, T>::~ExternalCheck()
```



```
 81: {
 82:     command.clear();
 83: }
 84:
 85: /*
 86:  * Check if the required tool is available
 87:  * @return true if the required tool has been found in the path,
 88:  * false otherwise.
 89:  */
 90: template<class Iterator, class T>
 91: bool ExternalCheck<Iterator, T>::available()
 92: {
 93:     return availability;
 94: }
 95:
 96: /*
 97:  * Executes the command in a new shell and return value
 98:  * @return the return value of the system call with "command" argument
 99:  * @pre The tool is supposed to be available before launching the run
100:  */
101: template<class Iterator, class T>
102: int ExternalCheck<Iterator, T>::run()
103: {
104:     const char * ccommand = command.c_str();
105:
106:     int ret = 0;
107:     if (system(NULL))
108:     {
109:         ret = system(ccommand);
110:     }
111:     else
112:     {
113:         cerr << "Could not find command interpreter on your system, sorry"
114:              << endl;
115:     }
116:     return ret;
117: }
118:
119: // ----------------------------------------------------------------------------
120: // Proto instanciations
121: // ----------------------------------------------------------------------------
122: #include <vector>
123:
124: /*
125:  * Instanciation du template ExternalCheck avec des <vector::const_iterator, int>
126:  */
127: template class ExternalCheck<std::vector<int>::const_iterator, int>;
128:
129: /*
130:  * Instanciation du template ExternalCheck avec des <vector::const_iterator, float>
131:  */
132: template class ExternalCheck<std::vector<float>::const_iterator, float>;
133:
134: /*
135:  * Instanciation du template ExternalCheck avec des <vector::const_iterator, float>
136:  */
137: template class ExternalCheck<std::vector<double>::const_iterator, double>;
```

```
/*
 * RCheck.h
 *
 *  Created on: 9 fÃ©vr. 2017
 *      Author: davidroussel
 */

#ifndef RCHECK_H_
#define RCHECK_H_

#include "ExternalCheck.h"

/**
 * R external checker
 */
template <class Iterator, class T>
class RCheck : public ExternalCheck<Iterator, T>
{
    /*
     * Reminder of ExternalCheck elements reused here
     */
    // using typename ExternalCheck<Iterator, T>::value_t;
    using typename ExternalCheck<Iterator, T>::diff_t;
    using ExternalCheck<Iterator, T>::tool;
    using ExternalCheck<Iterator, T>::availability;
    using ExternalCheck<Iterator, T>::command;

    public:
        /**
         * Protected constructor for R checker
         * @param baseDir the base directory to run the external tool from
         * @param begin an iterator to the first value of the tested sequence
         * @param end an iterator past the last value of the tested sequence
         * @param minValue minimum possible value in the sequence
         * @param maxValue maximum possible value in the sequence
         * @param meanValue computed mean value of the sequence
         * @param stdValue computed std value of the sequence
         * @param medianValue computed median value in the sequence
         */
        RCheck(const string & baseDir,
               const Iterator & begin,
               const Iterator & end,
               const T & minValue,
               const T & maxValue,
               const double & meanValue,
               const double & stdValue,
               const double & medianValue);
};

#endif /* RCHECK_H_ */
```



```
/*
 * RCheck.cpp
 *
 *  Created on: 9 fÃ©vr. 2017
 *      Author: davidroussel
 */

#include <sstream>  // For ostringstream
#include <algorithm>    // For copy algorihtm

#include "RCheck.h"

/*
 * Constructor for R checker
 * @param baseDir [not used] the base directory to run the external tool from
 * @param begin an iterator to the first value of the tested sequence
 * @param end an iterator past the last value of the tested sequence
 * @param minValue minimum possible value in the sequence
 * @param maxValue maximum possible value in the sequence
 * @param meanValue computed mean value of the sequence
 * @param stdValue computed std value of the sequence
 * @param medianValue computed median value in the sequence
 */
template<class Iterator, class T>
RCheck<Iterator, T>::RCheck(const string &,
                            const Iterator & begin,
                            const Iterator & end,
                            const T & minValue,
                            const T & maxValue,
                            const double & meanValue,
                            const double & stdValue,
                            const double & medianValue) :
    ExternalCheck<Iterator, T>("Rscript", "", begin, end, minValue, maxValue,
                               meanValue, stdValue, medianValue)
{

    ostringstream commandStream;

    commandStream << tool;
    // commandStream << "-e 'frame = read.csv(\"test.csv\", header = TRUE, sep = \";\")'";
    commandStream << " -e 'values = c(";
    diff_t limit = (end - begin);
    diff_t i;
    Iterator it;
    for (i = 0, it = begin; i < limit && it != end; ++i, ++it)
    {
        commandStream << *it;
        if (i < (limit - 1))
        {
            commandStream << ", ";
        }
    }
    commandStream << ")'";
    commandStream << " -e 'minValue = " << minValue << "'";
    commandStream << " -e 'maxValue = " << maxValue << "'";
    commandStream << " -e 'meanValue = " << meanValue << "'";
    commandStream << " -e 'stdValue = " << stdValue << "'";
    commandStream << " -e 'medianValue = " << medianValue << "'";
    commandStream << " -e 'source(\"check_TP_STL.R\")'";
    commandStream
        << " -e 'checkvalues(values, minValue, maxValue, meanValue, stdValue, medianValue)'";

    command = commandStream.str();
}

#include <vector>

/*
 * Instanciation du template ExternalCheck avec des <vector::const_iterator, int>
 */
template class RCheck<std::vector<int>::const_iterator, int> ;

/*
 * Instanciation du template ExternalCheck avec des <vector::const_iterator, float>
 */
template class RCheck<std::vector<float>::const_iterator, float> ;

/*
 * Instanciation du template ExternalCheck avec des <vector::const_iterator, float>
 */
```

```
81: template class RCheck<std::vector<double>::const_iterator, double> ;
```



```
 1: /*
 2:  * main.cpp
 3:  *
 4:  *  Created on: 21 fÃ©vr. 2013
 5:  *      Author: davidroussel
 6:  */
 7:
 8: #include <cstdlib>      // for EXIT_SUCCESS
 9: #include <unistd.h>     // for getcwd
10: #include <cmath>        // for fabs & sqrt
11: #include <vector>       // for vector
12: #include <iostream>     // for cout
13: #include <fstream>      // for ofstream
14: #include <algorithm>    // for all std algorithms
15: #include <iterator>     // for ostream_iterator
16: #include <sstream>      // for istringstream & ostringstream
17: #include <typeinfo>     // for typeid
18:
19: using namespace std;
20:
21: #include "BoundedRandomGenerator.h"
22: #include "StatFunctor.h"
23: #include "ExternalCheck.h"
24: #include "RCheck.h"
25:
26: /**
27:  * VÃ©rification de ... sur [first, last)
28:  * @tparam InputIterator un input itÃ©rateur (evt constant car on ne modifie
29:  * pas ce qui est pointÃ© par les itÃ©rateurs)
30:  * @param first l'itÃ©rateur sur le premier Ã©lÃ©ment
31:  * @param last l'itÃ©rateur au delÃ  du dernier Ã©lÃ©ment
32:  * @return vrai si ..., false sinon.
33:  */
34: template<class InputIterator>
35: bool mysteryCheck(const InputIterator & first, const InputIterator & last)
36: {
37:     if (first == last)
38:     {
39:         return true;
40:     }
41:
42:     InputIterator iter = first;
43:     InputIterator next = first;
44:
45:     while (++next != last)
46:     {
47:         if (*next < *iter)
48:         {
49:             return false;
50:         }
51:         ++iter;
52:     }
53:     return true;
54: }
55:
56: /**
57:  * Recherche de la mÃ©diane dans [first, last) triÃ©
58:  * @tparam RAIterator un random access iterator car on a besoin de
59:  * pouvoir utiliser l'arithmÃ©tique des itÃ©rateurs pour trouver le milieu de
60:  * [first, last)
61:  * @param first l'itÃ©rateur sur le premier Ã©lÃ©ment
62:  * @param last l'itÃ©rateur au delÃ  du dernier Ã©lÃ©ment
63:  * @return une paire d'itÃ©rateur indiquant la position du ou des elements qui
64:  * constituent la mÃ©diane :
65:  * Si first == last on renvoie une paire d'itÃ©rateur pointant sur <last, last>
66:  * Si [first, last) n'est pas triÃ© on renvoie une paire d'itÃ©rateur pointant sur
67:  * <last, last>
68:  * Soit n = last - first le nombre d'Ã©lÃ©ments
69:  *  - si n est impair : pair<...> contient deux fois le mÃªme itÃ©rateur pointant
70:  *      sur le ((n+1)/2)^iÃ¨me Ã©lÃ©ment
71:  *  - si n est pair : pair<...> contient
72:  *      - dans first un itÃ©rateur pointant sur le (n/2)^iÃ¨me Ã©lÃ©ment
73:  *      - dans second un itÃ©rateur pointant sur le ((n/2) + 1)^iÃ¨me Ã©lÃ©ment
74:  */
75: template<class RAIterator>
76: pair<RAIterator, RAIterator> median(const RAIterator & first,
77:                                     const RAIterator & last)
78: {
79:     using diff_t = typename iterator_traits<RAIterator>::difference_type;
80:     using category = typename iterator_traits<RAIterator>::iterator_category;
```

```
 81:
 82:     pair<RAIterator, RAIterator> p(last, last);
 83:
 84:     // TODO A complÃ©ter ...
 85:
 86:     return p;
 87: }
 88:
 89: /**
 90:  * Recherche de la mÃ©diane dans [first, last) NON triÃ©
 91:  * @tparam RAIterator un random access iterator car on a besoin de
 92:  * pouvoir utiliser l'arithmÃ©tique des itÃ©rateurs pour trouver le milieu de
 93:  * [first, last)
 94:  * @param first l'itÃ©rateur sur le premier Ã©lÃ©ment
 95:  * @param last l'itÃ©rateur au delÃ  du dernier Ã©lÃ©ment
 96:  * @return une paire d'itÃ©rateurs indiquant la position du ou des Ã©lÃ©ments qui
 97:  * constituent la mÃ©diane.
 98:  */
 99: template<class RAIterator>
100: pair<RAIterator, RAIterator> median2(const RAIterator & first,
101:                                      const RAIterator & last)
102: {
103:     using diff_t = typename iterator_traits<RAIterator>::difference_type;
104:     using value_t = typename iterator_traits<RAIterator>::value_type;
105:     using pointer = typename iterator_traits<RAIterator>::pointer;
106:     using category = typename iterator_traits<RAIterator>::iterator_category;
107:
108:     pair<RAIterator, RAIterator> p(last, last);
109:
110:     // TODO A complÃ©ter ...
111:
112:     return p;
113: }
114:
115: /**
116:  * Programme principal
117:  * @param argc nombre d'arguments
118:  * @param argv arguments
119:  * @return EXIT_SUCCESS si tout se passe bien, EXIT_FAILURE sinon
120:  */
121: int main(int argc, char * argv[])
122: {
123:     /*
124:      * Typedefs pour simplifier les dÃ©clarations de variables
125:      */
126:     using Element = double;
127:     using Container = vector<Element>;
128:     //  using iterator = Container::iterator;
129:     using const_iterator = Container::const_iterator;
130:     using Printer = ostream_iterator<Element>;
131:
132:     // nombre d'Ã©lÃ©ments par dÃ©faut
133:     const size_t DefaultNbElts = 10;
134:     const Element minVal = Element(-12.3);
135:     const Element maxVal = Element(25.7);
136:     const Element defaultVal = Element(0);
137:     size_t nbElts = DefaultNbElts;
138:     size_t buffSize = 256;
139:     char buff[buffSize];
140:     ofstream logStream;
141:     bool log = false;
142:     bool useExtrenalCheck = true;
143:
144:     /*
145:      * RÃ©cupÃ©ration du rÃ©pertoire courant pour les outils de vÃ©rif (matlab ou R)
146:      */
147:     getcwd(buff, buffSize);
148:     string baseDir(buff);
149:
150:     /*
151:      * S'il y a un argument au programme on recherche un nouveau nombre
152:      * d'Ã©lÃ©ments
153:      */
154:     if (argc > 1)
155:     {
156:         istringstream iss(argv[1]);
157:         iss >> nbElts;
158:         if (!iss || nbElts < 3)
159:         {
160:             nbElts = DefaultNbElts;
```



```
161:         }
162:     }
163:
164:     // conteneur utilisÃ© dans le programme
165:     Container v(nbElts, defaultVal);
166:
167:     // output iterator utilisÃ© pour afficher les valeurs d'une Ã©tendue
168:     // d'itÃ©rateurs sur la console
169:     Printer printer(cout, " ");
170:     // si le nombre d'Ã©lÃ©ments est trop grand, on redirige vers un fichier de log
171:     if (nbElts > 100)
172:     {
173:         log = true;
174:         // Affichage dans un fichier de log
175:         logStream.open("main.log", ofstream::out);
176:         printer = Printer(logStream, " ");
177:     }
178:
179:     // ============================================================================
180:     // remplissage de v avec des valeurs comprises entre -12.3 et +25.7
181:     // en utilisant un algo avec BoundedRandomGenerator
182:     // ============================================================================
183:
184:     // TODO Ã  complÃ©ter ...
185:
186:     // ============================================================================
187:     // Affichage de v
188:     // ============================================================================
189:     cout.precision(4); // affichage des nombres flottants avec 4 chiffres apv
190:     cout << "v = ";
191:     // TODO Ã  complÃ©ter ...
192:     cout << endl;
193:
194:     // ============================================================================
195:     // Calcul de la moyenne et de l'Ã©cart type de v en utilisant
196:     // StatFunctor dans un algorithme
197:     // ============================================================================
198:     StatFunctor<Element> stat; // TODO Ã  complÃ©ter ...
199:     cout << "moyenne des elts = " /* TODO << moyenne */ << " Â± " /* << TODO Ã©cart type */
200:         << endl;
201:
202:     // ============================================================================
203:     // Copie triÃ©e de v dans vt en utilisant un algorithme
204:     // ============================================================================
205:     Container vt(v.size()); // pour recevoir une copie triÃ©e de v
206:     // TODO Ã  complÃ©ter ...
207:
208:     cout << "v triÃ© : vt = ";
209:     // TODO Ã  complÃ©ter ...
210:     cout << endl;
211:
212:     // ============================================================================
213:     // Recherche d'un couple d'itÃ©rateurs sur la mÃ©diane de
214:     // [vt.begin(), vt.end())
215:     // ============================================================================
216:     /*
217:      * On vÃ©rifie que [vt.begin(), vt.end()) est triÃ©
218:      */
219:     bool sorted; // TODO Ã  complÃ©ter par sorted = XXXX(...);
220:     if (sorted)
221:     {
222:         cout << "vt est triÃ©";
223:     }
224:     else
225:     {
226:         cout << "vt n'est pas triÃ©";
227:     }
228:     cout << endl;
229:
230:     /*
231:      * On recherche la paire d'itÃ©rateurs correspondant Ã  la mÃ©diane dans vt
232:      * (triÃ©) en utilisant median<const_iterator>(...)
233:      */
234:     pair<const_iterator, const_iterator> sortedMedianItPair; // TODO Ã  complÃ©ter par = median(...)
235:
236:     /*
237:      * Affichage de la valeur de la mÃ©diane uniquement si vt Ã©tait non vide
238:      * et triÃ©:
239:      *  - si sortedMedianItPair.first == sortedMedianItPair.second
240:      *       une seule valeur : celle de *(sortedMedianItPair.first)
```

```
241:      *  - sinon
242:      *      deux valeurs : la valeur de la mÃ©diane est alors
243:      *      (*(sortedMedianItPair.first) + *(sortedMedianItPair.second)) / 2.0
244:      */
245:     double medianValue = 0.0;
246:
247:     cout << "La mÃ©diane de vt est = ";
248:     // TODO Ã  complÃ©ter ...
249:     cout << endl;
250:
251:     // ======================================================================
252:     // Recherche d'un itÃ©rateur sur la mÃ©diane dans [v.begin(), v.end())
253:     // ======================================================================
254:     /*
255:      * On recherche la paire d'itÃ©rateurs correspondant Ã  la mÃ©diane dans v
256:      * (non triÃ©)
257:      */
258:     pair<const_iterator, const_iterator> medianItPair; // TODO Ã  complÃ©ter par = median2(...);
259:
260:     /*
261:      * Affichage de la valeur de la mÃ©diane uniquement si vt Ã©tait non vide
262:      * et triÃ©:
263:      */
264:     cout << "La mÃ©diane de v est = ";
265:     // TODO Ã  complÃ©ter ...
266:     cout << endl;
267:
268:     // ======================================================================
269:     // VÃ©rification des min, max, moyenne, std et mediane en utilisant matlab
270:     // ======================================================================
271:     /*
272:      * TODO dÃ©commentez les lignes ci-dessous lorsque StatFunctor est prÃªt et
273:      * que vous disposez d'une valeur pour medianValue.
274:      * Si Matlab et/ou R sont installÃ©s, ils sont utilisÃ©s pour vÃ©rifier les
275:      * valeurs de
276:      *  - contenu de v / [min,max]
277:      *  - mean
278:      *  - std
279:      *  - median
280:      */
281: //  if (useExtrenalCheck)
282: //  {
283: //      ExternalCheck<const_iterator, Element> * checker =
284: //          new RCheck<const_iterator, Element>(baseDir,
285: //                                              v.cbegin(),
286: //                                              v.cend(),
287: //                                              minVal,
288: //                                              maxVal,
289: //                                              stat.mean(),
290: //                                              stat.std(),
291: //                                              medianValue);
292: //      if (checker->available())
293: //      {
294: //          checker->run();
295: //      }
296: //
297: //      delete checker;
298: //  }
299:
300:     // ======================================================================
301:     // VÃ©rification de la propriÃ©tÃ©
302:     // (abs(mediane - moyenne) / ecarttype) <= sqrt (3/5)
303:     // ======================================================================
304:     double test;  // TODO Ã  complÃ©ter ...
305:     double testMax = sqrt(3.0/5.0);
306:     cout << test << " ";
307:     if (test <= testMax)
308:     {
309:         cout << "<= ";
310:     }
311:     else
312:     {
313:         cout << "> ";
314:     }
315:     cout << testMax << endl;
316:
317:     // ======================================================================
318:     // Soustraction de la (moyenne des elts de v) Ã  v
319:     // en utilisant uniquement des algorithmes et de foncteurs standards
320:     // ======================================================================
```



```
321:     /*
322:      * Soustraction de la moyenne de v Ã  tous les elts de v
323:      */
324:     // TODO Ã  complÃ©ter ...
325:
326:     /*
327:      * Afichage de v modifiÃ©
328:      */
329:     cout << "v - " /*<< TODO  moyenne */ << " = ";
330:     // TODO Ã  complÃ©ter ...
331:     cout << endl;
332:
333:     // ======================================================================
334:     // Copie des elts >= 0 de v dans vp et < 0 dans vn
335:     // en utilisant uniquement des algorithmes et de foncteurs standards
336:     // Attention ! vp & vn n'ont pas de taille prÃ©alable, il faudra donc
337:     // explicitement insÃ©rer les Ã©lÃ©ments dans vp & vn avec des itÃ©rateurs de
338:     // sortie particuliers destinÃ©s Ã  insÃ©rer.
339:     // ======================================================================
340:     Container vp;
341:     Container vn;
342:
343:     /*
344:      * Copie des elts de v >= 0 dans vp
345:      */
346:     // TODO Ã  complÃ©ter ...
347:
348:     /*
349:      * Copie des elts de v < 0 dans vn
350:      */
351:     // TODO Ã  complÃ©ter ...
352:
353:     /*
354:      * Affichage des elts de vp
355:      */
356:     cout << "elts >= 0 de v : ";
357:     // TODO Ã  complÃ©ter ...
358:     cout << endl;
359:
360:     /*
361:      * Affichage des elts de vn
362:      */
363:     cout << "elts < 0 de v : ";
364:     // TODO Ã  complÃ©ter ...
365:     cout << endl;
366:
367:     /* ======================================================================
368:      * Peut on remplacer "vector" par "deque" au dÃ©but du programme ?
369:      * TODO DÃ©veloppez la rÃ©ponse :
370:      */
371:
372:     /* ======================================================================
373:      * Peut on remplacer "vector" par "list" au dÃ©but du programme ?
374:      * TODO DÃ©veloppez la rÃ©ponse :
375:      */
376:
377:     /* ======================================================================
378:      * Quelles sont les conditions pour que l'on puisse affecter Ã  Element
379:      * un type quelconque (des CompteBanque par exemple)
380:      * TODO Listez les opÃ©rations nÃ©cessaires
381:      *  -
382:      */
383:
384:     if (log)
385:     {
386:         logStream.close();
387:     }
388:
389:     return EXIT_SUCCESS;
390: }
```