```
/*
-----------------------------------------------------------------------------
Laboratory  : labo_04
File        : labo_04_carvalho_bruno_gallay_david.cpp
Author(s)   : Carvalho Bruno et Gallay David
Date        :

Purpose     : Prove the good working of classes defined in others files.
Remark(s)   :
                There is the github repository:

Compiler    : MinGW-g++ 6.3.0 and g++ 7.4.0
-----------------------------------------------------------------------------*/
#include <iostream>
#include <string>
#include <cstdlib>
#include <stdexcept>
#include "src/temps.h"

using namespace std;

#define WAIT_ENTER while(cin.get()!='\n')

int main() {
    Temps<int> t1(1, 1, 1);
    Temps<float> t2(0, 0, 0);

    Temps<double> t3(1, 23, 45);

    std::cout << t1 << std::endl;
    std::cout << t2 << std::endl;
    std::cout << t3 << std::endl;
    std::cout << t3.asHeure() << std::endl;
    std::cout << t3.asMinute() << std::endl;
    std::cout << (Temps<int>)t2 << std::endl;
    std::cout << t1 + t1 << std::endl;
    std::cout << t1 + t2 << std::endl;
    std::cout << (float)t3 << std::endl;
    std::cout << (double)t3 << std::endl;
    std::cout << (long long)t3 << std::endl;
    std::cout << std::string(t3) << std::endl;
    try {
        t2 - t3;
    } catch (std::exception& e) {
        std::cerr << e.what() << std::endl;
    }

    cout << "Please, press <ENTER> to end the program" << endl;
    WAIT_ENTER;
    return EXIT_SUCCESS;
}

```

```
/*
-----------------------------------------------------------------------------------
Laboratory  : labo_04
File        : temps.h
Author(s)   : Carvalho Bruno et Gallay David
Date        :

Purpose     : Declaring function Temps
Remark(s)   : since we can't assume which type will be used
              and that we don't assert types requirements,
              no functions could be qualified as noexcept
              (neiter the getters)

                There is the github repository:
                https://github.com/dgheig/Ba2-inf2-labo04

Compiler    : g++ 7.4.0
-----------------------------------------------------------------------------------*/

#ifndef TEMPS_H
#define TEMPS_H
#include <iostream>
#include <string>

template <typename T>
class Temps {
    public:
        /**
         * @brief constructor of class Temps
         * @param heure
         * @param minute
         * @param seconde
         * @throw negative_value resulting time must be positive
         */
        Temps(T heure=0, T minute=0, T seconde=0);

        /**
         * @brief constructor by copy
         * @param temps
         */
        template<typename U>
        Temps(const Temps<U>& temps);

        /**
         * @brief constructor by copy
         * @param temps
         */
        Temps(const Temps& temps);

        /**
         * @brief construct a full time HH:MM:SS just by converting seconds
         * @param seconde
         * @throw negative_value resulting time must be positive
         */
        Temps(T seconde);

        /**
         * @brief getter for hours
         * @return heure
         */
        T getHeure() const;

        /**
         * @brief getter for minutes
         * @return minute
         */
        T getMinute() const;

        /**
         * @brief getter for seconds
         * @return seconde
         */
        T getSeconde() const;

        /**
         * @brief convert into hours representation
```

```
77            * @return hours
78            */
79           T asHeure() const;
80
81           /**
82            * @brief convert into minutes representation
83            * @return minutes
84            */
85           T asMinute() const;
86
87           /**
88            * @brief convert into seconds representation
89            * @return seconds
90            */
91           T asSeconde() const;
92
93           /**
94            * @brief print object representation to stream
95            * @param stream
96            * @return reference on the stream parameter
97            */
98           std::ostream& print(std::ostream& stream=std::cout) const;
99
100          /**
101           * @brief allow to modify value of hours
102           * @param heure
103           * @throw negative_value hours must be positives
104           */
105          void setHeure(T heure);
106
107          /**
108           * @brief allow to modify value of minutes
109           * @param minute
110           * @throw std::invalid_argument minutes must be in [0, 60[
111           */
112          void setMinute(T minute);
113
114          /**
115           * @brief allow to modify value of seconds
116           * @param seconde
117           * @throw std::invalid_argument seconds must be in [0, 60[
118           */
119          void setSeconde(T seconde);
120
121          /**
122           * @brief surcharge of operator =, allow us to copy an object Temps into an other
123           * @param temps
124           * @return reference on itself
125           */
126          template<typename U>
127          Temps& operator=(const Temps<U>& temps);
128
129          /**
130           * @brief surcharge of operator =, allow us to copy an object Temps into an other
131           * @param temps
132           * @return reference on itself
133           */
134          Temps& operator=(const Temps& temps);
135
136          /**
137           * @brief surcharge of operator +=, allow us to addition object
138           * @param temps
139           * @return reference on itself
140           */
141          Temps& operator+=(const Temps& temps);
142
143          /**
144           * @brief surcharge of operator -=, allow us to substract object
145           * @param temps
146           * @return reference on itself
147           */
148          Temps& operator-=(const Temps& temps);
149
150          /**
151           * @brief surcharge of operator +=, allow us to addition object
152           * @param temps
```

```
153             * @return reference on itself
154             */
155            template<typename U>
156            Temps& operator+=(const Temps<U>& temps);
157
158            /**
159             * @brief surcharge of operator -=, allow us to substract object
160             * @param temps
161             * @return reference on itself
162             */
163            template<typename U>
164            Temps& operator-=(const Temps<U>& temps);
165
166            /**
167             * @brief surcharge of operator ==, allow us to compare two object
168             * @param temps
169             * @return true if objects are equal
170             */
171            bool operator==(const Temps& temps) const;
172
173            /**
174             * @brief surcharge of operator !=, allow us to compare two object
175             * @param temps
176             * @return true if objects are different
177             */
178            bool operator!=(const Temps& temps) const;
179
180            /**
181             * @brief cast object into float, it means it convert an HH:MM:SS into h.xxxxx
182             * @return casted value to float
183             */
184            operator float() const;
185
186            /**
187             * @brief cast object into double, it means it convert an HH:MM:SS into h.xxxx
188             * @return casted value to double
189             */
190            operator double() const;
191
192            /**
193             * @brief cast object into long double, it means it convert HH:MM:SS into h.xxxx
194             * @return casted value to long double
195             */
196            operator long double() const;
197
198            /**
199             * @brief cast object into long long int, convert our full time HH:MM:SS into seconds
200             * @return casted value to long long
201             */
202            operator long long() const;
203
204            /**
205             * @brief cast object into string, allow us to print it into HH:MM:SS format
206             * @return casted value to std::string
207             */
208            operator std::string() const;
209
210        private:
211            static T toSeconde(T heure, T minute, T seconde);
212            void fromSeconde(T seconde);
213            T _heure;
214            T _minute;
215            T _seconde;
216    };
217
218
219    /**
220     * @brief return the difference without check for overflow
221     *        Object is granted to be positiv
222     * @param t1 instance of Temps
223     * @param t2 instance of Temps
224     * @return return the difference between t1 and t2.
225     */
226    template <typename T1, typename T2>
227    Temps<T1> operator+(Temps<T1> t1, const Temps<T2>& t2);
228
```

```
229  /**
230   * @brief return the difference without check for underflow
231   *        checks could be done using operator<
232   *        Object is granted to be positiv
233   * @param t1 instance of Temps
234   * @param t2 instance of Temps
235   * @return return the difference between t1 and t2.
236   */
237  template <typename T1, typename T2>
238  Temps<T1> operator-(Temps<T1> t1, const Temps<T2>& t2);
239
240  /**
241   * @param t1 instance of Temps
242   * @param t2 instance of Temps
243   * @return true if t1 is smaller than t2
244   */
245  template <typename T1, typename T2>
246  Temps<T1> operator<(Temps<T1> t1, const Temps<T2>& t2);
247
248  /**
249   * @param t1 instance of Temps
250   * @param t2 instance of Temps
251   * @return true if t1 is bigger than t2
252   */
253  template <typename T1, typename T2>
254  Temps<T1> operator>(Temps<T1> t1, const Temps<T2>& t2);
255
256  /**
257   * @brief print temps object representation to stream
258   * @param stream
259   * @return reference on the stream parameter
260   */
261  template <typename T>
262  std::ostream& operator<<(std::ostream& stream, Temps<T> temps);
263
264
265  #include "temps.ipp"
266  #endif // TEMPS_H
267
```

```
/*
-------------------------------------------------------------------------------
Laboratory  : labo_04
File        : temps.ipp
Author(s)   : Carvalho Bruno et Gallay David
Date        :

Purpose     : Defining function Temps
Remark(s)   :
                There is the github repository:
                https://github.com/dgheig/Ba2-inf2-labo04

Compiler    : g++ 7.4.0
-------------------------------------------------------------------------------*/

#ifndef TEMPS_IPP
#define TEMPS_IPP

#define SEC_IN_MIN 60
#define MIN_IN_H 60
#define SEC_IN_H (SEC_IN_MIN * MIN_IN_H)

#include <sstream>
#include "exceptions.h"

// Constructors

template<typename T>
Temps<T>::Temps(T heure, T minute, T seconde): _heure(0), _minute(0), _seconde(0) {
    // Let the user enter negative parameter
    // as long as the resulting time is positive
    fromSeconde(toSeconde(heure, minute, seconde));
}


template<typename T>
template<typename U>
Temps<T>::Temps(const Temps<U>& temps): Temps() {
    *this = temps;
}

template<typename T>
Temps<T>::Temps(const Temps<T>& temps): Temps() {
    *this = temps;
}

template <typename T>
Temps<T>::Temps(T seconde): Temps() {
    fromSeconde(seconde);
}

template <typename T>
void Temps<T>::fromSeconde(T seconde) {
    if (seconde < 0)
        throw negative_value("La classe Temps ne peut avoir une valeur negative");
    _heure = (T)(int)(seconde / SEC_IN_H);
    seconde -= _heure * SEC_IN_H;
    _minute = (T)(int)(seconde / SEC_IN_MIN);
    seconde -= _minute * SEC_IN_MIN;
    _seconde = seconde;
}

// Getters
template<typename T>
T Temps<T>::getHeure() const {
    return _heure;
}

template<typename T>
T Temps<T>::getMinute() const {
    return _minute;
}

template<typename T>
T Temps<T>::getSeconde() const {
    return _seconde;
```

```
 77  }
 78
 79  template<typename T>
 80  T Temps<T>::asHeure() const {
 81      return asSeconde() / SEC_IN_H;
 82  }
 83
 84  template<typename T>
 85  T Temps<T>::asMinute() const {
 86      return asSeconde() / SEC_IN_MIN;
 87  }
 88
 89  template<typename T>
 90  T Temps<T>::asSeconde() const {
 91      return toSeconde(
 92          getHeure(),
 93          getMinute(),
 94          getSeconde()
 95      );
 96  }
 97
 98  template<typename T>
 99  std::ostream& Temps<T>::print(std::ostream& stream) const {
100      return stream << _heure << ':' << _minute << ':' << _seconde;
101  }
102
103  // Setters
104  template<typename T>
105  void Temps<T>::setHeure(T heure) {
106      if (heure < 0)
107          throw negative_value("La classe Temps ne peut avoir d'heures negatives");
108      _heure = heure;
109  }
110
111  template<typename T>
112  void Temps<T>::setMinute(T minute) {
113      if (minute < 0 or MIN_IN_H <= minute)
114          throw std::invalid_argument("Les minutes doivent etre entre 0 et 59");
115      _minute = minute;
116  }
117
118  template<typename T>
119  void Temps<T>::setSeconde(T seconde) {
120      if (seconde < 0 or SEC_IN_MIN <= seconde)
121          throw std::invalid_argument("Les secondes doivent etre entre 0 et 59");
122      _seconde = seconde;
123  }
124
125  // Operators
126
127  template<typename T>
128  template<typename U>
129  Temps<T>& Temps<T>::operator=(const Temps<U>& temps) {
130      _heure = (T)temps.getHeure();
131      _minute = (T)temps.getMinute();
132      _seconde = (T)temps.getSeconde();
133      return *this;
134  }
135
136  template<typename T>
137  Temps<T>& Temps<T>::operator=(const Temps<T>& temps) {
138      _heure = temps._heure;
139      _minute = temps._minute;
140      _seconde = temps._seconde;
141      return *this;
142  }
143
144  template<typename T>
145  bool Temps<T>::operator!=(const Temps<T>& temps) const {
146      return (_heure != temps._heure) || (_minute != temps._minute) || (_seconde !=
      temps._seconde);
147  }
148
149  template<typename T>
150  bool Temps<T>::operator==(const Temps<T>& temps) const {
151      return !(this != temps);
```

```
152   }
153
154   template<typename T>
155   Temps<T>& Temps<T>::operator+=(const Temps<T>& temps) {
156       fromSeconde(asSeconde() + temps.asSeconde());
157       return *this;
158   }
159
160   template<typename T>
161   Temps<T>& Temps<T>::operator-=(const Temps<T>& temps) {
162       fromSeconde(asSeconde() - temps.asSeconde());
163       return *this;
164   }
165
166   template<typename T>
167   T Temps<T>::toSeconde(T heure, T minute, T seconde) {
168       return heure * SEC_IN_H + minute * SEC_IN_MIN + seconde;
169   }
170
171   template<typename T>
172   template<typename U>
173   Temps<T>& Temps<T>::operator+=(const Temps<U>& temps) {
174       return *this += Temps<T>(temps);
175   }
176
177   template<typename T>
178   template<typename U>
179   Temps<T>& Temps<T>::operator-=(const Temps<U>& temps) {
180       return *this -= Temps<T>(temps);
181   }
182
183   template <typename T1, typename T2>
184   Temps<T1> operator+(Temps<T1> temps1, const Temps<T2>& temps2) {
185       return temps1 += temps2;
186   }
187
188   template <typename T1, typename T2>
189   Temps<T1> operator-(Temps<T1> temps1, const Temps<T2>& temps2) {
190       return temps1 -= temps2;
191   }
192
193   template <typename T1, typename T2>
194   Temps<T1> operator<(Temps<T1> t1, const Temps<T2>& t2) {
195       return t1.asSeconde() < t2.asSeconde();
196   }
197   template <typename T1, typename T2>
198   Temps<T1> operator>(Temps<T1> t1, const Temps<T2>& t2) {
199       return t2 < t1;
200   }
201
202   template<typename T>
203   std::ostream& operator<<(std::ostream& stream, Temps<T> temps) {
204       return temps.print(stream);
205   }
206
207   template<typename T>
208   Temps<T>::operator float() const {
209       return Temps<float>(*this).asHeure();
210   }
211
212   template<typename T>
213   Temps<T>::operator double() const {
214       return Temps<double>(*this).asHeure();
215   }
216
217   template<typename T>
218   Temps<T>::operator long double() const {
219       return Temps<long double>(*this).asHeure();
220   }
221
222   template<typename T>
223   Temps<T>::operator long long() const {
224       return (long long)asSeconde();
225   }
226
227   template<typename T>
```

```
228   Temps<T>::operator std::string() const {
229       std::stringstream ss;
230       print(ss);
231       return ss.str();
232   }
233
234   #endif // TEMPS_IPP
```

```
/*
-----------------------------------------------------------------------------
Laboratory  : labo_04
File        : exceptions.h
Author(s)   : Carvalho Bruno et Gallay David
Date        :

Purpose     : Defining customs exceptions
Remark(s)   : We did not define what() function overload
              since we use std::invalid_argument's one

              In my opinion, defining this class is a theorical exercise
              std::invalid_argument was more than good enough for all the needs
              of this project

                There is the github repository:
                https://github.com/dgheig/Ba2-inf2-labo04

Compiler    : g++ 7.4.0
-----------------------------------------------------------------------------*/

#ifndef EXCEPTIONS_H
#define EXCEPTIONS_H

#include <stdexcept>
#include <string>

class negative_value: public std::invalid_argument {
    public:
        /**
         * @brief Constructor
         * @param msg error message
         */
        negative_value(const char* msg): std::invalid_argument(msg) {}

        /**
         * @brief Constructor
         * @param msg error message
         */
        negative_value(std::string msg): negative_value(msg.c_str()) {}
};


#endif // EXCEPTIONS_H
```