```
/*
ThermistorNTC.h - Library to used to derive a precise temperature of a thermistor,
fastest Calc (26~18% faster)
v0.2

Copyright © 2021 Francisco Rafael Reyes Carmona.
All rights reserved.

rafael.reyes.carmona@gmail.com


  This file is part of ThermistorNTC.

  ThermistorNTC is free software: you can redistribute it and/or modify
  it under the terms of the GNU General Public License as published by
  the Free Software Foundation, either version 3 of the License, or
  (at your option) any later version.

  ThermistorNTC is distributed in the hope that it will be useful,
  but WITHOUT ANY WARRANTY; without even the implied warranty of
  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
  GNU General Public License for more details.

  You should have received a copy of the GNU General Public License
  along with ThermistorNTC.  If not, see <https://www.gnu.org/licenses/>.

*/

#if ARDUINO >= 100
  #include "Arduino.h"
#else
  #include "WProgram.h"
#endif

#ifndef ThermistorNTC_h
#define ThermistorNTC_h

enum Thermistor_connection {
  VCC,
  GND
  };


class Thermistor {
    private:
      #if defined(__LGT8F__)
        int _ADC_MAX = 4096;  //ADC max. value (4093) + 1 -> 4096. But it will be 4069 by design.
      #elif defined(__AVR_ATmega168__) || defined(__AVR_ATmega328P__) ||
      defined(__AVR_ATmega328__)
```

```
      defined(__AVR_ATmega328__)
        int _ADC_MAX = 1024;  //ADC max. value (1023) + 1 -> 1024.
      #else
        int _ADC_MAX = 1024;  //ADC max. value (1023) + 1 -> 1024.
      #endif
        int _PIN;
        long _RESISTOR = 10000L;
        long _NTC_25C = 0L;
        double _A = 0.0;
        double _B = 0.0;
        double _C = 0.0;
        double _D = 0.0;
        float _BETA = 0.0;
        float _VREF;

        float _alphaEMA_LOW = 0.91;

        void calcCoefficients3(float, long, float, long, float, long);
        void calcCoefficients4(float, long, float, long, float, long,
        float, long);
        double calcNTC(Thermistor_connection ConType = VCC);
        float getADC(int numsamples = 15);
        void SteinhartHart(Thermistor_connection ConType = VCC);
        void SteinhartHart_beta(Thermistor_connection ConType = VCC);
        void SteinhartHart_fast(Thermistor_connection ConType = VCC);

    public:
        double _temp_k;
        double _temp_c;

        Thermistor() = delete; // Constructor por defecto.
        Thermistor(int, long, long, double, double, double, double,
        float); // Constructor para 4 parametros (A,B,C,D).
        Thermistor(int, long, long, double, double, double, float); //
        Constructor para 3 parametros (A,B,D.. C = 0).
        Thermistor(int, long, long, float, float); // Constructor para
        parametro BETA del termistor.
        Thermistor(int, long, long, float, long, float, long, float,
        float); // Constructor cuando se desconoce los parámetros del
        termistor. 3 Coeficientes
        Thermistor(int, long, long, float, long, float, long, float, long,
        float, float); // Constructor cuando se desconoce los parámetros
        del termistor. 4 Coeficientes
        Thermistor(const Thermistor&) = delete; // Constructor de copia.

        void setADC(int);
        void setEMA(float);

        double getTempKelvin(Thermistor_connection ConType = VCC);
        double getTempCelsius(Thermistor_connection ConType = VCC);
        double getTempFahrenheit(Thermistor_connection ConType = VCC);
```

```
91
92          double fastTempKelvin(Thermistor_connection ConType = VCC);
93          double fastTempCelsius(Thermistor_connection ConType = VCC);
94          double fastTempFahrenheit(Thermistor_connection ConType = VCC);
95
96          double getTempKelvin_SteinHart(Thermistor_connection ConType =
            VCC);
97          double getTempCelsius_SteinHart(Thermistor_connection ConType =
            VCC);
98          double getTempFahrenheit_SteinHart(Thermistor_connection ConType =
            VCC);
99  };
100
101 #endif
102
```