```c
/**
 * @file Sensors.h
 * @brief Hardware abstraction for Sensors
 * Copyright (C) 2017 Ethan Wells
 *
 * This program 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.
 *
 * This program is distributed in the hope that it will be useful, but WITHOUT
 * ANY WARRANTY; without even the implied warranty ofMERCHANTABILITY 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 this program. If not, see <https://www.gnu.org/licenses/>
 */

#ifndef CARL_SENSORS_H_
#define CARL_SENSORS_H_

#include "API.h"

/**
 * The different types of Sensors
 */
typedef enum {
        /** Analog Sensor */
        Analog,

        /** High Resolution Analog */
        AnalogHR,

        /** Digital Sensor */
        Digital,

        /** Quadrature shaft encoder */
        Quad,

        /** Ultrasonic Sensor */
        Sonic,

        /** Gyro Sensor */
        Gyroscope,
```

```
        /** Integrated Motor Encoder */
        IME,

        /** Placeholder for a late init Sensor */
        Placeholder,
} SensorType;

/**
 * A struct representing a Sensor of a given type
 */
typedef struct Sensor {
        /** Child in the linked list */
        struct Sensor *child;

        /** Current Sensor value */
        int value;

        /** Current Sensor's velocity */
        int velocity;

        /** The average of the Sensor value and it's child's value */
        int averageVal;

        /** The average velocity if the Sensor's velocity and it's children's velocity */
        int averageVel;

        /** Recalculation function of the Sensor's value */
        float (*recalc)(int);

        /** Whether or not the Sensor's value is inverted */
        bool inverted;

        /** Sensor port */
        unsigned char port;

        /** Calibration data, like a gyro multiplier. Can also be used as a bool */
        unsigned short calibrate;

        int        zero;
        SensorType _type;
        void      *_pros;
        Mutex      _mutex;
        unsigned long _lastUpdate;
        int _lastValue;
} Sensor;
```

```
/**
 * Refresh the information on the Sensor
 *
 * @param s the Sensor to refresh
 */
void   sensorRefresh(Sensor *s);

/**
 * Reset a sensor's value
 *
 * @param the Sensor to reset
 */
void   sensorReset(Sensor *s);

/**
 * Create a new Sensor
 *
 * @param type      the type of SensorType, either a Digital, Analog,
 * AnalogHR, Quad, Sonic, or Gyroscope
 * @param port      the port in which the Sensor in in
 * @param inverted  whether or not to invert the value
 * @param calibrate the calibration value in some cases, or anything but 0 to
 * calibrate the Sensor object
 *
 * @return the new Sensor
 */
Sensor newSensor(SensorType     type,
                 unsigned char  port,
                 bool           inverted,
                 unsigned short calibrate);

/**
 * Create a new digital Sensor
 *
 * @param port     the port that the digital Sensor is in
 * @param inverted whether or not to invert the value
 *
 * @return the new digital Sensor object
 */
Sensor newDigital(unsigned char port,
                  bool          inverted);

/**
 * Create a Sonic (aka ultrasonic) Sensor
 *
 * @param orange the port that the orange cable is in
```

```
 * @param yellow the port that the yellow cable is in
 *
 * @return the new ultrasonic Sensor object
 */
Sensor newSonic(unsigned char orange,
                unsigned char yellow);

/**
 * Create and initialize a quadrature encoder (the red ones)
 * @param top      the port that the top wire on the encoder is in
 * @param bottom   the port that the bottom wire on the encoder is in
 * @param inverted whether or not the Sensor's value should be inverted
 *
 * @return the new quadrature encoder Sensor object
 */
Sensor newQuad(unsigned char top,
               unsigned char bottom,
               bool          inverted);

/**
 * Create a new analog Sensor
 *
 * @param port      the port that the Sensor is in
 * @param calibrate whether or not to calibrate the sensor
 *
 * @return the new analog Sensor object
 */
Sensor newAnalog(unsigned char port,
                 bool          calibrate);

/**
 * Create a new analog HR sensor
 *
 * @param port the port that the Sensor is in
 *
 * @return the new analog Sensor object with High Resolution
 */
Sensor newAnalogHR(unsigned char port);

/**
 * Create a gyroscope Sensor
 *
 * @param port        the analog port that the gyro is plugged into
 * @param inverted    whether or not the gyroscope is inverted
 * @param calibration the calibration of the Sensor
 *
```

```
 * @return the new gyro Sensor object
 */
Sensor newGyro(unsigned char port,
               bool          inverted,
               int           calibration);

/**
 * Create a IME Sensor
 *
 * @param num         the number that the IME is on the daisy chain
 * @param inverted    whether or not it's inverted
 *
 * @return the new IME Sensor object
 */
Sensor newIME(unsigned char num,
              bool          inverted);

#endif // CARL_SENSORS_H_
```