Интересный метод реализации классов датчиков на C++

Что требуется от датчика?

Датчик представляет из себя устройство, которое позволяет преобразовать некоторые сигналы в цифровой сигнал, понятный микроконтроллеру. В конечном итоге (при программировании микроконтроллера) этот сигнал при считывании с датчика сохраняется в памяти для дальнейшей обработки.

Пример класса для чтения датчика

```
class Sensor
{
  private:
    int sensorValue;

  public:
    Sensor(int pin)
    {
        pinMode(pin, INPUT);
    }
    int update()
    {
      sensorValue = analogRead(pin);
      return sensorValue;
    }
    int getValue()
    {
      return sensorValue;
    }
};
```

PROF

Такой способ реализации чтения датчика прост и понятен. Мы имеем возможность централизованно обновить значение датчика и с помощью геттера можно безопасно получить его значение.

Однако, я хотел бы поделиться с моей точки зрения более простым и интересным способом реализации чтения датчика.

Использование константной ссылки

Что нам по сути нужно от класса для чтения датчика? Нужно иметь возможность прочитать новое значение, сохранить его для дальнейших обращений и иметь возможность его прочитать. Для этого нам необходимо иметь поле для хранения значения датчика и метод для его чтения. Однако,

чтение значения с помощью геттера несколько громоздко и требует написания кода геттера, что может быстро надоесть. Код однообразный и громоздкий.

Можно ли от него избавится? В принципе да: сделав поле публичным.

```
class Sensor
{
  public:
    int sensorValue;

    Sensor(int pin)
    {
        pinMode(pin, INPUT);
    }
    int update()
    {
      sensorValue = analogRead(pin);
      return sensorValue;
    }
    int getValue()
    {
      return sensorValue;
    }
};
```

Но это очень плохая идея. Это значение может кто угодно изменить, что очень небезопасно.

Но, это не единственный способ избавиться от геттера! В C++ есть такая волшебная вещь, как ссылка. По сути это просто второе название для переменной, которое можно использовать равноценно оригинальной переменной. НО!

У ссылки на переменную есть возможность добавить дополнительные модификаторы, а именно нас интересует модификатор `const`. Пример кода:

PROF

```
void illustration()
{
    int x = 10; // Создаем переменную x
    const int& y = x; // Создаем константую ссылку на переменную x
    x = 20; // Переменная x может быть свободно изменена
    Serial.println(y); // При этом все изменения видны в ссылке y тоже
    y = 30; // [ERROR] НО! Через ссылку y изменить значение переменной x
НЕЛЬЗЯ!
}
```

Таким образом мы можем спокойно объявить приватное поле для хранения значения датчика и обновлять его когда нам необходимо, а для чтения создать публичную константую ссылку на это поле. Вот так:

```
class Sensor
{
private:
    int sensorValue;

public:
    const int& q_sensorValue = sensorValue;

    Sensor(int pin)
    {
        pinMode(pin, INPUT);
    }
    int update()
    {
      sensorValue = analogRead(pin);
      return sensorValue;
    }
};
```

> Комментарий. Преффикс `q` означает, что это выходной сигнал. Нотация идет из ПЛК (программируемых логических контроллеров), где каждый регистр имеет один из трех преффиксов: `M` - для внутренних переменных, `I` - для входных сигналов и `Q` - для выходных сигналов.

Таким образом мы избавились от геттера и получили более безопасный способ реализации чтения датчика. При этом, чтение значения по ссылке происходит просто обращением к памяти без вызова дополнительных функций.

```
Sensor sensor(A0);

sensor.update();
Serial.println(sensor.q_sensorValue); // Выведет значение датчика
sensor.q_sensorValue = 100; // [ERROR]
```

PROF

Такая вот интересная штука!