Writing a GPS library

The SparkFun GNSS library is very well written. However, it contains a lot of extra functionality and configuration. We are going to write a library that uses the SparkFun one, but only exposes what we need.

Let's start by making "GPS.h".

```
1   #ifndef GPS_H
2   #define GPS_H
3
4   #include <SparkFun_u-blox_GNSS_v3.h>
5
6   // The main data that we care to extract from the GPS
7   struct GPSData
8   {
9       long lattitude;
10      long longitude;
11      byte SIV;      // satellites in view
12      long heading; // Relative to magnetic north
13      long speed;   // Ground Speed
14  };
15
16  class GPS
17  {
18      private:
19          SFE_UBLOX_GNSS_SERIAL myGNSS;
20      public:
21          void GetData(GPSData &data);
22          bool setupGPS(Stream &theSerial);
23          GPS();
24  };
25
26  #endif
```

The struct, you can think of it as Class that only has variables associated with it. No methods.

Another way is to think of it as a grour of variables ~~that can be acces~~ that are related and can be accessed with a common name.

Writing GPS Library

GPS.cpp:

```cpp
#include "gps.h"

GPS::GPS(){}

bool GPS::setupGPS(Stream &theSerial)
{
    if (!myGNSS.begin(theSerial))
    {
        return false;
    }

    // Set some settings to reduce noise on the communication lines
    myGNSS.setUART1Output(COM_TYPE_UBX);
    myGNSS.setI2COutput(COM_TYPE_UBX);
    myGNSS.saveConfiguration();
    return true;
}

void GPS::GetData(GPSData &data)
{
    // Here, we have passed the GPSData struct by reference.
    // So we fill it up with the new data and it is accessible
    // using that same object.
    data.lattitude = myGNSS.getLatitude();
    data.longitude = myGNSS.getLongitude();
    data.SIV       = myGNSS.getSIV();
    data.heading   = myGNSS.getHeading();
    data.speed     = myGNSS.getGroundSpeed();
}
```

On setup we pass in a reference to the Serial port we are using. That is what the "&" is saying. When we get the data, we also pass in a reference to the memory where we want the variables to go.

Example Driver Code

```
#include "src/GPS/gps.h"

// Create a GPS Object and data holder
GPS myGPS = GPS();
GPSData myGPSData;

void printGPSData(GPSData &theGPSData)
{
    // Here we are passing the GPSData by reference
    Serial.print(theGPSData.lattitude);
    Serial.print(" ");
    Serial.print(theGPSData.longitude);
    Serial.print(" ");
    Serial.print(theGPSData.SIV);
    Serial.print(" ");
    Serial.print(theGPSData.heading);
    Serial.print(" ");
    Serial.print(theGPSData.speed);
    Serial.println();
}
void setup()
{
    // Serial monitor for output.
    Serial.begin(9600);
    while(!Serial); // Wait for serial monitor to open.
    Serial.println("#GetObsessed!");

    // Tell the micro controller that
    // the gps is connected to Serial1.
    Serial1.begin(38400);
    if (!myGPS.setupGPS(Serial1))
    {
        // If the setup fails, we freeze.
        digitalWrite(LED_BUILTIN, HIGH);
        Serial.println("GPS failed to begin.");
        while(1);
    }
}
void loop()
{
    // Tell the GPS to put GPSData inside
    // the myGPSData struct.
    myGPS.GetData(myGPSData);

    // Print out all the data
    printGPSData(myGPSData);
}
```

Example Output:

```
334679730 -819909447 6 0 74
334679640 -819909470 6 0 161
334679705 -819909473 6 0 88
334679716 -819909480 6 0 164
334679726 -819909555 6 0 15
334679730 -819909561 6 0 18
334679728 -819909553 6 0 40
334679729 -819909542 6 0 81
334679728 -819909410 11 0 36
```