Summary:

This activity will walk students through stacking and unstacking of the Adafruit Ultimate GPS Logger Shield (henceforth referred to as the Shield) and interfacing it with the Arduino Mega. By the end of this activity, students will know how to properly stack/unstack the Shield, have an understanding of the difference between direct connect and software serial connect, and will be able to parse raw NMEA sentences.

Materials:

Each student should have the following materials, equipment and supplies:

- Computer with Arduino IDE installed
- USB-AB programming cable
- Arduino Mega microcontroller
- Assembled Adafruit Ultimate GPS Logger Shield
    - Includes: GPS Logger Shield, coin cell battery, and 2 female-male jumpers
- Lock Ring Spreader Pliers

Procedure:

Activity A: Stacking/Unstacking a Shield

When a shield is attached to an Arduino, it is called stacking. Stacking and unstacking is not complicated, but it can result in bent pins if done improperly. Always be aware of the Shield's pins and do not force movement.

1. Get the Shield and the Mega. Make sure the Mega is disconnected from power.

   Steps 2 – 3 explain how to stack the Shield. Always be aware of the header pins and if they are at risk of bending.

2. Align the Shield with the Mega, as shown in Figure 1. Do not push down on anything yet. Just ensure the header pins align with the proper Mega pins.

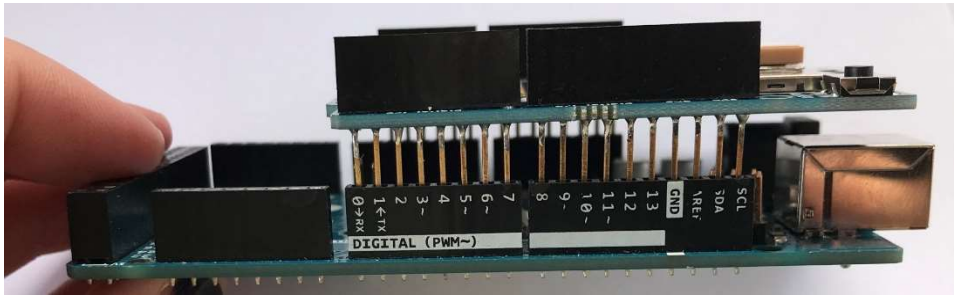

Figure 1: Shown is how to align the Shield with the Mega. The microSD slot is on the side closest to you.

3. Gently (do not shove it) push the Shield onto the Mega by applying even pressure. It is okay to rock the Shield slightly, but do not bend the header pins. Continue until the Shield is fully seated on the Mega; it should look like Figure 2.

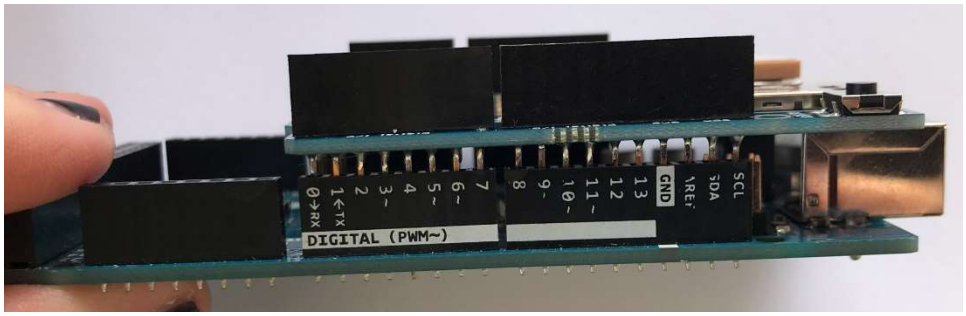

Figure 2: Shown is the Shield fully attached to the Mega. Part of the header pins are still visible. The Shield has a little bit of space between itself and the programming jack.

Steps 4 – 7 explain how to unstack the Shield and the Mega. Always be aware of the header pins and their risk of being bent.

4. The tool shown in Figure 3 is a lock ring spreader pliers. It can be used to unstack the Shield and the Mega while reducing the chances of pins bending. Open and close it a couple of times to get a feel for the instrument

Figure 3: This is the tool you will use to destack the Shield and the Mega. Using this lock ring spreader pliers will help prevent pins from being bent.

5. Stick the pliers in between the Shield and the Mega as shown in Figure 4. Gently push the Shield up just a little. Do not push it so far off that the pins on the other side of the Shield bend.

6. Move the pliers to the other position shown in Figure 4. Again, gently push the Shield up just a little.

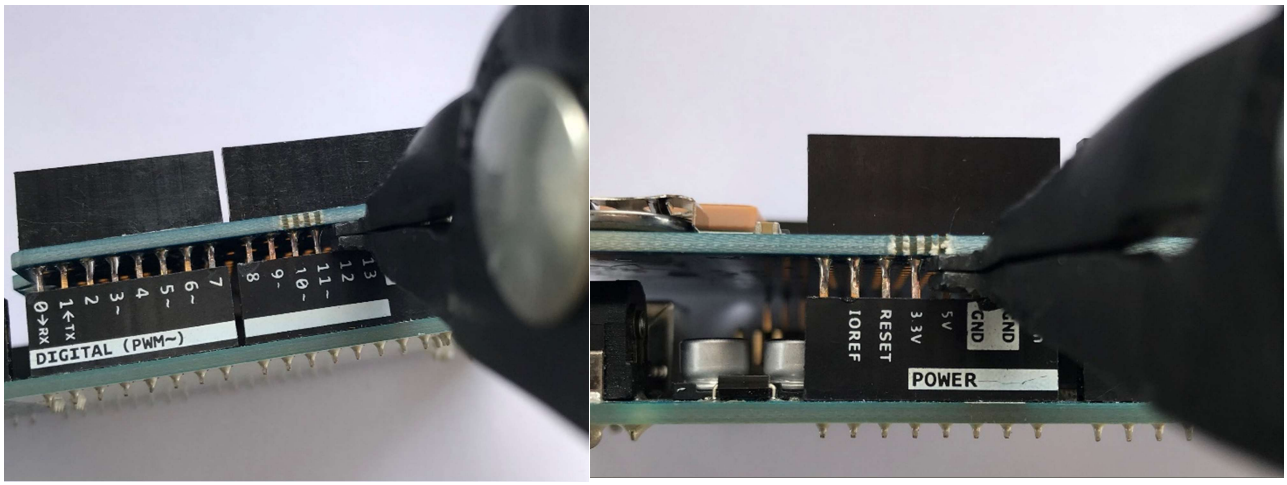

Figure 4: Shown are two of the places the pliers can be placed to separate the Mega from the Shield. Do no fully separate the pieces in one attempt. Go back and forth between these two positions at least three times to ensure that pins do not get bent while separating.

7. Repeat steps 5 and 6 until the Shield can just be lifted off.

Activity B: Downloading Arduino Libraries

1. In order to program the GPS, you will need to download the Adafruit GPS library. This library can be found at https://github.com/adafruit/Adafruit_GPS. Go to this library and download it.
    a. It is currently version 1.2

2. This folder will download as a zip folder. Extract it.

3. Open the extracted folder. Inside, there will be another folder named "Adafruit_GPS-master." Rename it as "Adafruit_GPS."

4. Move the "Adafruit_GPS" folder to the libraries folder for the Arduino IDE.
    a. This should be found at Documents >> Arduino >> libraries.

5. In order to save data to the SD card using the Arduino Mega, the SD library will be required. It can be found at https://github.com/adafruit/SD. Download it.
    a. It is currently version 3.

6. Use the same process that was used for the GPS library to extract the folder. Don't forget to rename the folder from "SD-master" to "SD." Move the extracted SD folder to your library folder.

7. Restart the Arduino IDE if it is currently open.

Activity C: Direct Connect

1. If not already on the Mega, put the Shield on the Mega. Make sure the switch on the Shield is flipped to Soft Serial. This will allow a sketch to be uploaded on the Mega.

2. Create a new, blank sketch in the Arduino IDE. Upload this sketch to the Mega and open the Serial Monitor.
    a. Make sure the baud rate in the Serial Monitor is set to 9600 baud.

3. After uploading a blank sketch, it is now safe to flip the switch from "Soft Serial" to "Direct." This bypasses the Arduino chip so the Shield can communicate directly with the computer. Do this now.

4. Raw GPS data should start appearing in the Serial Monitor. These are the NMEA sentences talked about in Lecture 18 and Appendix J – NMEA Strings. Figure 5 shows an example of what your Serial Monitor might look like.

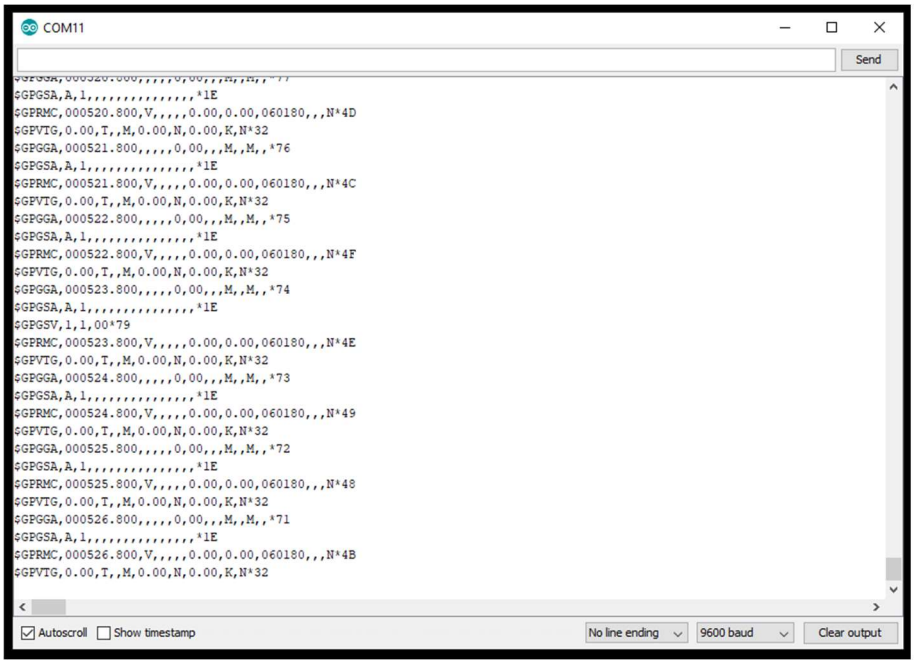

Figure 5: Shown is an example of what raw data from the Shield looks like using Direct Connect. This screenshot was taken when the Shield did not have a fix. If your Shield has a fix, there will be more information instead of blank data fields (the consecutive commas are blank data fields).

5. If you don't have a fix, as in Figure 5, try moving the Mega and Shield closer to a window. If it is cloudy outside, a fix may not be achievable. The red LED on the Shield will blink once every second if it does not have a fix. It will blink once every 15 seconds if it does have a fix, and the raw GPS data will look more like Figure 6.

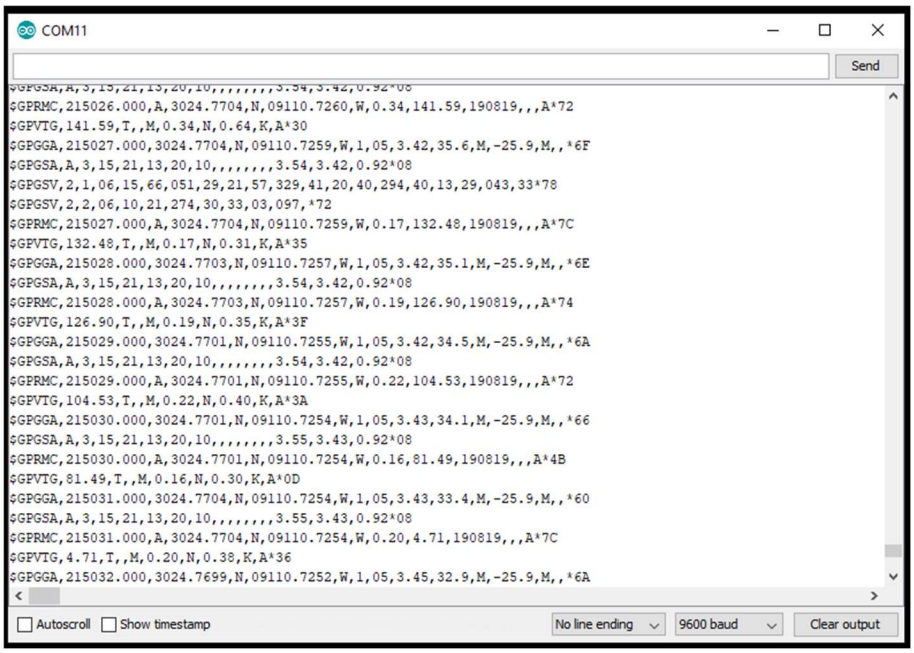

Figure 6: Shown is an example of what raw data from the Shield looks like using Direct Connect. This screenshot was taken when the Shield had a fix. When compared with Figure 5, more data fields are filled.

6. Before sending commands to the Shield using the Serial Monitor, make sure that the "Both NL & CR" is selected next to the baud rate.

7. Below is a table of GPS commands and what they do. To send a command, type it into the Serial Monitor. Experiment with all of these commands to achieve different outputs from the Shield. Reference Appendix J – NMEA Strings to get more information regarding specific NMEA strings.
    a. After sending a command, a response will be sent from the Shield along the lines of "$PMTK001,314,3*36." This is the antenna status of the Shield.

Table 1: GPS Commands

Command	Command Result
$PMTK314,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGLL sentence
$PMTK314,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPRMC sentence
$PMTK314,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPVTG sentence
$PMTK314,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGGA sentence
$PMTK314,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGSA sentence
$PMTK314,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0*29	Turn on only the GPGSV sentence
$PMTK314,0,1,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on GPRMC and GPGGA sentences
$PMTK314,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn on all NMEA sentence output
$PMTK314,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0*28	Turn off all NMEA sentence output

8. Before proceeding, show an instructor how to send a few commands using direct connect.

Activity D: Software Serial Connect

1. Let's change to communicating with the GPS Logger Shield using Software Serial. First, disconnect the Arduino Mega from the computer. Flip the switch on the Shield so that it is selecting Soft Serial.

2. Take the remaining two right angle header pins and insert them into the Mega's pins 18 and 19 – TX1 and RX1, respectively.

3. Connect the Shield RX to the Mega TX1 and the Shield TX to the Mega RX1 using the female-to-female jumpers and the right angle headers. The Shield and Arduino Mega should look similar to Figure 7.

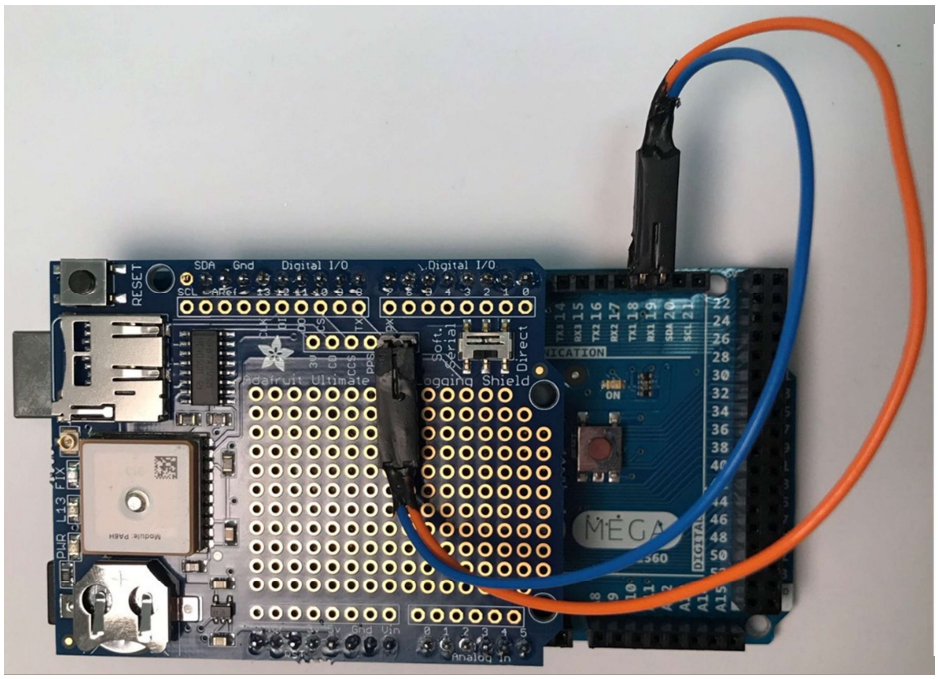

Figure 7: Shown is the Software Serial connection for the Shield and the Mega. The breakout pins RX and TX should be connected to the Mega pins TX1 and RX1, respectively. If these connections are switched, no data will be received from the GPS unit.

4. Next, open up a new sketch in the Arduino IDE and save it as *GPSSoftwareSerialIntroduction.io*.

5. First, we need to include the library that is needed to communicate with the GPS. This is the Adafruit_GPS library. Figure 8 shows how to include it.

```
24 /* Include statements for required libraries ***********************************************/
25 #include <Adafruit_GPS.h>     // Library for using the Adafruit Ultimate GPS Logger
```

Figure 8: This is how a library, specifically the GPS library, is included. This allows the user to access the library's functions

6. Figure 9 defines the pins that will be used for the GPS communication and connecting the hardware port to the GPS. We jumpered the breakout TX and RX to the Mega's Serial 1 (TX1 and RX1), so that is the port that we define as GPSSerial.

```
/* Adafruit Ultimate GPS Logger Shield ************************************************/
#define GPSSerial Serial1     // Communicate with GPS using Seriall (RX and TX 1)
Adafruit_GPS GPS(&GPSSerial); // Connect to the GPS on the hardware port
```

Figure 9: This code defines the GPS Serial port.

7. The proper way to get GPS data is by using an interrupt. This means we do not actively query the GPS serial communication to read that data. Instead, a new character is automatically read every millisecond. We will create the interrupt function after we create the setup() function. Before setup(), include the code from Figure 10. These lines will allow us to use the interrupt.

```
/* Other global variables *************************************************************/
boolean usingInterrupt = false;    // Flag for if GPS interrupt should be enabled
void useInterrupt(boolean);        // Define function to enable/disable interrupt
```

Figure 10: This code allows the use of an interrupt (the interrupt function will be defined later) to read the GPS. setup() should finish before the interrupt is enabled, so usingInterrupt is originally defined as being false

8. In setup(), begin the two serial communications: the Serial Monitor communication and the GPS communication.
    a. The Serial Monitor baud rate should be set to 115200. This will allow the Mega to print to the Serial Monitor without missing GPS data.
    b. The GPS baud rate should be set to 9600 (a standard for GPS communication). Start this communication by with GPS.begin(9600);.

9. Figure 11 shows code that tells the GPS what types of information should be received and the rate at which the information should be sent.
    a. This link https://github.com/adafruit/Adafruit_GPS/blob/master/Adafruit_GPS.h is the documentation for the Adafruit GPS library. It defines all the commands that can be sent to the GPS. It also defines the different information that can be parsed.

b. When the GPS sends data, it does a data dump. The data received sits in the serial buffer until it is read by the Arduino Mega. When the command to read a byte from the GPS is run, it reads a byte from this buffer.

```
45  /* Define data being requested from the GPS and at what rate **********/
46  GPS.sendCommand(PMTK_Q_RELEASE);                  // Send the GPS module version
47  GPS.sendCommand(PGCMD_NOANTENNA);                 // Turn off antenna status updates
48  //GPS.sendCommand(PGCMD_ANTENNA);                   // Turn on antenna status updates
49  GPS.sendCommand(PMTK_SET_NMEA_UPDATE_1HZ);        // Set GPS update rate
50
51  // These define what NMEA sentence(s) will be received. Uncomment one
52  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GLLONLY);
53  GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCONLY);
54  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_VTGONLY);
55  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GGAONLY);
56  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSAONLY);
57  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_GSVONLY);
58  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_RMCGGA);
59  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_ALLDATA);
60  //GPS.sendCommand(PMTK_SET_NMEA_OUTPUT_OFF);
```

Figure 11: This code defines what information will be sent by the GPS and the rate at which the information will be sent. This will have the GPS send the module version, no antenna status, and only the RMC NMEA string. It will send updated information (i.e. a RMC sentence) every 1 second.

10. Below the commands from Figure 11, enable the interrupt that will read the GPS serial buffer. Do this by calling the useInterrupt() function and inputting true.

11. Below setup(), define the interrupt service routine, as in Figure 12. When enabled, it reads one character from the GPS serial port.
    a. This interrupt will be called using an internal timer, Timer0. Timer0 pulses every millisecond. This means that the interrupt is going to run every millisecond.

```
66 SIGNAL(TIMER0_COMPA_vect) {
67 /******* Interrupt Service Routine ************************************************
68  *  This is the interrupt service routine. It reads a character from the GPS.
69  ******************************************************************************/
70   char c = GPS.read();      // Read a character from the GPS
71 }
```

Figure 12: This is the function that actually reads a character from the GPS serial buffer. It operates using Timer0, an internal timer on the Mega. Using Timer0 means that this function will run every millisecond when the interrupt is enabled.

12. Next, define the function that enables/disables the interrupt. Figure 13 shows this code. This function sets the value of the usingInterrupt flag. If the interrupt is to be enabled, this flag is true. If the interrupt is to be disabled, this flag is false.

```
void useInterrupt(boolean v) {
/******* useInterrupt ********************************************************************
 *  This function is enables or disables the SIGNAL(TIMER0_COMPA_vect) interrupt.
 ********************************************************************************************/
  // millis() uses Timer0, so we'll interrupt in the middle and call the function above
  if (v) {
    OCR0A = 0xAF;
    TIMSK0 |= _BV(OCIE0A);
    usingInterrupt = true;
  }

  // Do not call the interrupt function COMPA anymore
  else {
    TIMSK0 &= ~_BV(OCIE0A);
    usingInterrupt = false;
  }
}
```

Figure 13: This function enables or disables the interrupt. If it is passed true, it enables the interrupt. If it is passed false, it disables it.

13. The main loop should just be an if statement. If a new NMEA sentence is received, print the sentence to the Serial Monitor. Figure 14 shows how loop() should look.

```
void loop() {
/******* loop ********************************************************************************
 *  This function runs repeatedly. It prints new NMEA strings to the Serial Monitor
 ********************************************************************************************/
  // If a new NMEA sentence is received, print it on the Serial Monitor
  if (GPS.newNMEAreceived()) Serial.print(GPS.lastNMEA());
}
```

Figure 14: This is main loop. Every time it runs, it checks if a new NMEA sentence has been fully received. If it has, then the sentence is printed to the Serial Monitor.

14. Plug in the Arduino Mega and upload the sketch. Open the Serial Monitor and set the baud rate to 115200.

15. Play around with different GPS commands. Make sure to try every command from Figure 11 at least once and understand the information that is being printed out on the Serial Monitor.

16. Using the GPS library documentation, try commands to change the update rate. Does the new output in the Serial Monitor make sense?

17. Before proceeding, show an instructor the different commands and explain the difference between direct connect and software serial.

Activity E: Parsing Data Introduction

1. It is often beneficial to parse NMEA sentences to extract specific information. This activity will show how to use the GPS library to parse information automatically. Begin by opening the sketch created in Activity D. Save a new copy of this sketch as *GPSParsingIntroduction.io*.

2. Edit setup() so the commands being sent are the same as in Figure 11.

3. Figure 15 shows the command to parse a NMEA sentence, Line 99. If a parse is successful, the sentence "Parse was successful!" and the sentence will be printed to the Serial Monitor.
    a. The command GPS.parse(GPS.lastNMEA()) returns a Boolean. If the parse was successful, it returns true. If it failed, it returns false. Because of this, it is beneficial to use the command as the condition of an if statement. If the parse is successful, then the if statement runs.

```
91  void loop() {
92  /******* loop ****************************************************************************
93   *  This function runs repeatedly. It prints new NMEA strings to the Serial Monitor.
94   ******************************************************************************************/
95    // Check if a new NEMA sentence has been received
96    if (GPS.newNMEAreceived()) {
97
98      // Try to parse the NMEA sentence.  If it can't be parsed, exit the if statement.
99      if (GPS.parse(GPS.lastNMEA())) {
100       Serial.println("Parse was successful!");
101       Serial.print(GPS.lastNMEA());
102     }
103   }
104 }
```

Figure 15: For the goal of printing "Parse was successful!" and the NMEA sentence, this is an example of how the loop() function could look. This is not the only way to achieve this end result.

4. The different types of information that are available after parsing can be found at https://github.com/adafruit/Adafruit_GPS/blob/master/src/Adafruit_GPS.h . Go to this link and familiarize yourself with the different information that can be obtained.
    a. The information that can come from parsing starts at "`uint8_t hour,`" which is towards the end of the page.

5. Let's have loop() print out the current seconds from the parsed information. Beneath Line 101 in Figure 15, add the lines *Serial.print("The current seconds are: ");* and *Serial.println(GPS.seconds);*. Upload the sketch.

6. Edit the loop() function so that it prints out the following to the Serial Monitor after a successful parse:

    Parse was successful!
    Current time is HH:MM:SS

where HH:MM:SS is the current UTC time. Show an instructor before proceeding.

a. UTC stands for Coordinated Universal Time. It will not be the same as your time. Depending on daylight saving, UTC will be 5 or 6 hours ahead of you.

7. Edit setup() so that all data is being sent by the GPS module.

8. Edit the sketch so that every 5 seconds, the current time is printed to the Serial Monitor as well as the GPS fix quality and number of satellites being used for the GPS fix. With the exception of the time, make sure that all the information is labelled.
   a. Hint: It will be beneficial to use variables and if statements.
   b. Hint: The function millis().

9. Every 7 seconds print out the full GLL NMEA string and position information (altitude, longitude, and latitude).

10. Every 13 seconds, print out the speed and current date. Include a label for the speed.

11. Do not proceed until you have shown your instructor your code and the Serial Monitor.

Note: Looking at the GPS library documentation and comparing it to the information in NEMA reference, it is clear that the GPS library does not parse all NMEA sentences. The Adafruit_GPS.cpp file (https://github.com/adafruit/Adafruit_GPS/blob/master/src/Adafruit_GPS.cpp) shows that the only NMEA sentences being parsed are the RMC, GGA, and GLL sentences. Does this mean that the other three NMEA sentences are useless? No. They just need to be parsed manually. If you would like to do this, there is an extra activity in the reference folder. This is not required.