Please write code to solve one problem of your choosing and write up a short high level summary of how you would solve a second problem of your choosing.

Do not hesitate to ask clarifying questions from Kyle (kyle@flypyka.com) or Nathan (nathan@flypyka)!

Question 1: Determine the altitude of the airplane

Problem: Our aircraft have two downward-facing laser altimeters (which report the airplane's height above ground in meters) and a GPS (which reports the plane's altitude relative to mean sea level). When we are flying down low (below ~15 meters), we care very much exactly how far above the ground the aircraft is, in order to maintain a precise height for spraying and landing. As the plane gets higher, it tends to fly over rougher terrain, and it becomes more important to fly a smooth trajectory through space than to precisely follow the contour of the ground below. Also, due to dust, sensor noise, and other phenomena, one or both of the altimeters will sometimes report height readings that are not reflective of the plane's actual height above the ground. These incorrect readings will be physically impossible (e.g., they will jump around a lot between successive readings, 10ms apart). The naive approach of just using the data from the altimeters is clearly not sufficient to produce an accurate, smooth estimate of the plane's current height above the ground.

Your task is to write a C++ program that reads in the real airplane log data we've provided (as CSV files) of both correct and incorrect altimeter behavior and outputs (to standard out) a running estimate of the plane's true height above ground level. This estimate should be as accurate as possible given the data and should ideally be relatively smooth.

Note: You may assume that the data from the GPS is correct, but you cannot assume that the ground is perfectly flat even below 15 meters.

Note: It may be helpful to graph data during this. A relatively easy way to do this is with Python using Jupyter Notebook. You can easily read in data using pandas.read_csv() and plot it using matplotlib. This StackOverflow question explains the basics.

Note: Don't worry about exactly what size of feature is accepted vs rejected in your solution–that could easily be tuned either way.

Log Data: Altimeter Data Note: altimeter and GPS values are in meters, and successive time stamps are 0.010 seconds apart.

Question 2: Read a value from an ADC

Problem: You have an ATmega328PB microcontroller (datasheet) that's controlling a half-bridge for a DC motor controller. Your task is to write the C++ firmware for the microcontroller so that it can control the motor via PWM and measure/report the motor current to a host computer.

Note: Getting a full toolchain set up isn't worth the effort for this, so don't worry about simple syntax errors and stuff that would be easily caught by a compiler.

Microcontroller Setup:

- The uC fuses have been programmed so that it's operating from its internal RC oscillator at 8MHz with no clock divider.
- The uC is running on a 5V supply.
- The half-bridge is connected to GPIO pin PD6.
- The current sensor is connected to ADC0, and provides an analog voltage proportional to its measured current: 0 volts for 0 amps, and 5 volts for 100 amps.
- The uC does not have a hardware floating point unit.
- UART0 is connected to the host computer and is magically set up.

uC -> Host Communication Protocol:

- The uC should send an exponential moving average of the motor current draw to the host at 10Hz. The current should be encoded in amps as a single byte. E.g., if the average current draw is 50 amps, the uC should write the single byte 0x32 to the UART.

Requirements:

- The uC should output 50% duty-cycle Phase-Correct PWM using Timer0 on pin PD6 at ~1kHz.
- The uC should perform an ADC conversion at the center of each PWM frame (see Timer/Counter Overflow interrupt (TOV0)).
- The running average of the ADC conversion (converted to amps) should be sent over UART0 to the host processor at a rate of ~10Hz, using the UARTWrite function provided in the starter code.

Starter Code:

```
#include <stdint.h>

// Writes a single byte to UART0 to send it to the host computer. Pretend
the
// UART is initialized and this function is implemented elsewhere.
void UARTWrite(const uint8_t byte);

// Interrupt vector that will be called when the ADC interrupt is enabled
and an
// ADC conversion is completed.
void ADCInterruptVector() {}

// Entry point
int main() {
  while (1)
    ;
}
```

Question 3: Determine if a path intersects with an obstacle

Problem: You are given a path and an obstacle in 3d space. Your task is to write a function that determines if the path intersects the obstacle.

The XY component of the path will be a clockwise circular arc, defined by a center, radius, start and end angles (in degrees). The height of the path is a linear interpolation from a start height to an end height.
The obstacle is a convex polygon with a fixed height. It is encoded as an object consisting of an std::vector of the polygon vertices (in clockwise order) in the XY plane, and a height.

Note: The angles are referenced clockwise from the +y axis, so the clockwise arc from 0 to 90 degrees is the top right quarter of the full circle.

Note: The function does not have to be perfectly precise (i.e. an analytical solution).

Note: Eigen may be a useful library here, but don't feel obligated to use it–Feel free to change the objects below to represent the 2d coordinates differently.

Object structures:

```
#include <Eigen/Dense>
#include <vector>

class Path {
  const Eigen::Vector2d _center;
  const float _radius;
  const float _start_angle;
  const float _end_angle;
  const float _start_height;
  const float _end_height;

 public:
  Path(const Eigen::Vector2d& center, float radius, float start_angle, float end_angle,
       float start_height, float end_height)
      : _center(center),
        _radius(radius),
        _start_angle(start_angle),
        _end_angle(end_angle),
        _start_height(start_height),
        _end_height(end_height){};
};
```

```
struct Obstacle {
  const std::vector<Eigen::Vector2d> vertices;
  const float height;
  Obstacle(const std::vector<Eigen::Vector2d>& vertices, float height)
      : vertices(vertices), height(height){};
};
```

Some Test Cases:

```
Path path_1({0, 0}, 15, 0, 90, 25, 15);
Path path_2({10, 10}, 5, -45, 225, 18, 30);
Path path_3({10, -10}, 15, -15, 15, 60, 10);

Obstacle obstacle_1(
    std::vector<Eigen::Vector2d>({{10, 10}, {10, 0}, {0, 10}}), 30);
Obstacle obstacle_2(
    std::vector<Eigen::Vector2d>({{10, 0}, {10, 10}, {20, 10}, {20, 0}}), 20);
Obstacle obstacle_3(
    std::vector<Eigen::Vector2d>({{0, 10}, {0, 20}, {10, 20}, {10, 10}}), 20);
```

path_1 intersects obstacle_2, but not obstacle_1 and obstacle_3.
path_2 intersects obstacle_1 and obstacle_3, but not obstacle_2.
path_3 intersects obstacle_2, but not obstacle_1 and obstacle_3.