float sal_float;
float tds_float;
char *sg;
char *ec;

//default I2C ID number for EZO EC Circuit.
define address 100

//enable I2C.
//hardware initialization.

//float var used to hold the float value of the salinity.
//float var used to hold the float value of the conductivity.

//char pointer used in string parsing.
//char pointer used in string parsing.

//used to change the delay needed depending on the command sent to the EZO Class EC Circuit.

//we make a 48 byte character array to hold incoming data from the EC circuit.

//used to hold the I2C response code.

//a flag to signal when data has been received from the pc/mac/other.

//we need to know how many characters have been received.

//we make a 20 byte character array to hold incoming data from a pc/mac/other.

//enable I2C.

//we now print each value we parsed separately.

//this is the salinity value.

//we now print each value we parsed separately.

//we now print each value we parsed separately.

//let's pars the string at each comma.
//this function will break up the CSV string into its 4 individual parts.

//uncomment this function if you would like to break up the comma
//reset the serial event flag.
//print the data.
//exits the switch case.
//means there is no further data to send.
//decimal 254.
//exits the switch case.
//means the command has failed.
//decimal 2.
//reset the counter i to 0.

//if any other command has been sent we wait only 300ms.

//if a command has been sent to calibrate or take a
//reset the counter i to 0.

//if a character in the array is uppercase we change it to lowercase.
//we need to check each character in the array.

//we need to know how many characters have been received.

//the main loop.

//how many characters have been received.
//(pc/mac/other) until we see a <CR>. We also count
//the first byte is the response code, we read this separately.

//call the circuit and request 48 bytes (this is more than we need)
//wait the correct amount of time for the circuit to complete its instruction.

//wait the correct amount of time and request data.

//end the I2C data transmission.

//load this byte into our array.

//receive a byte.

//the first byte is the response code, we read this separately.

//call the circuit and request 48 bytes (this is more than we need)
//wait the correct amount of time for the circuit to complete its instruction.

//wait the correct amount of time and request data.

//end the I2C data transmission.

//load this byte into our array.

//how many characters have been received.
//(pc/mac/other) until we see a <CR>. We also count
//the first byte is the response code, we read this separately.

//call the circuit and request 48 bytes (this is more than we need)
//wait the correct amount of time for the circuit to complete its instruction.