Arduino Programming
Part 5: Functions Redux and Intro to Arrays

EAS 199B, Winter 2013

Gerald Recktenwald
Portland State University
gerry@me.pdx.edu

Goals

Review operation of the Salinity Sensor
* Motivation for turning power on and off
* Circuit for salinity sensor

Create functions for reading the conductivity sensor
* Only one function is needed (only one used at a time)
* Different functions have different features
* Change input data handling by using different functions
* Main program stays largely unchanged

Introduction to arrays
* Use arrays to store readings
* Compute average and standard deviation of the readings


Measuring salinity

Principle of operation
* Ions migrate to electrodes
* Ions exchange electrons with probes, causing current flow.
* Na+ is a spectator ion.
* Ion concentrations increase at electrodes when power is left on.
* Therefore, only turn on power during the time when reading is made. Leave it off otherwise.
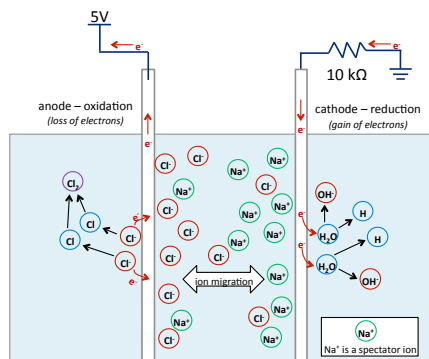
Measuring salinity

Sensor circuit

- ❖ It's a voltage divider
- ❖ Resistance decreases as salt concentration increases
- ❖ Voltage across fixed resistor increases when sensor resistance decreases, i.e. when salt concentration increases

Digital output

salinty sensor
(variable resistance)

Analog input

10 kΩ
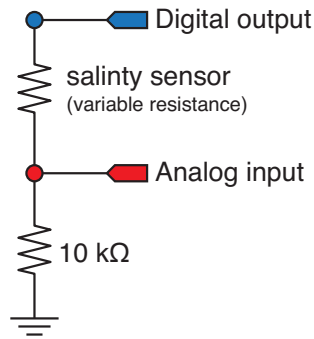
Study Questions

What is the voltage on the input pin for each of these conditions:

- ❖ If the electrical resistance of the water is zero?
- ❖ If the electrical resistance of the water is 10kΩ?
- ❖ If the electrical resistance of the water is ∞?

What is the input reading for each of those conditions?

IF the resistance varied linearly with salinity, would the voltage vary linearly with salinity?

Digital output

salinty sensor
(variable resistance)

Analog input

10 kΩ
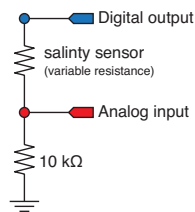
Programs for Reading the Salinity Sensor

1. Read one value at a time
   - ❖ Encapsulate the code in a function so it can be reused
2. Read multiple values and return an average
   - ❖ Code in a new function
3. Read multiple values and return average and standard deviation
   - ❖ Yet another function
   - ❖ Use an array to store readings, then compute statistics
   - ❖ Returning two values requires pointers

All three programs use the same circuit

Measuring salinity

Measurement algorithm

❖ Turn on the power with digital output to supply 5V to the voltage divider

❖ Wait for voltage transient to settle

❖ Read the voltage across fixed resistor

❖ Turn off the power

Digital output

salinty sensor
(variable resistance)

Analog input

10 kΩ
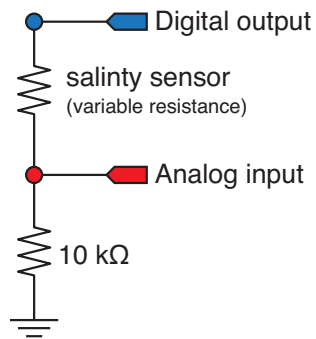
## Single reading of conductivity sensor

```
int power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int input_pin = 2; //  Analog input pin
  int reading;

  digitalWrite( power_pin, HIGH );   //  Turn on sensor
  delay(100);                        //  wait to settle
  reading = analogRead( input_pin ); //  Measure voltage
  digitalWrite( power_pin, LOW );    //  Turn off power

  Serial.println(reading);
}
```
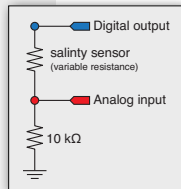
Create a function to read the sensor

Why use functions?

❖ Code in the loop function is just high level commands

▸ Overall logic is easier to read and change

▸ Reduce likelihood of error as overall code logic changes

❖ Keep details of sensor-reading contained in the function

▸ Variables defined in the function are "local"

▸ Details can change, e.g. to increase speed or reduce memory usage without changing the logic of the main function.

▸ Reuse the code in other projects: build a library of reusable components

Use a function to make a single reading

```
int salinity_power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int salinity_input_pin = 2; //  Analog input pin
  int salinity;

  salinity = salinity_reading( salinity_power_pin, salinity_input_pin );
  Serial.println(salinity);
}
// --------------------------------------------------------------
int salinity_reading( int power_pin, int input_pin ) {

  int reading;

  digitalWrite( power_pin, HIGH );    //  Turn on the sensor
  delay(100);                         //  Wait to settle
  reading = analogRead( input_pin );  //  Read voltage
  digitalWrite( power_pin, LOW );     //  Turn off the sensor
  return reading;
}
```

Encapsulate single reading in a function

```
int salinity_power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int salinity_input_pin = 2; //  Analog input pin
  int salinity;

  salinity = salinity_reading( salinity_power_pin, salinity_input_pin );
  Serial.println(salinity);
}
// ---------------------------------------------------------
int salinity_reading( int power_pin, int input_pin ) {

  int reading;

  digitalWrite( power_pin, HIGH );    //  Turn on the sensor
  delay(100);                         //  Wait to settle
  reading = analogRead( input_pin );  //  Read voltage
  digitalWrite( power_pin, LOW );     //  Turn off the sensor
  return reading;
}
```

Local variables power_pin and input_pin exist only inside salinity_reading

Encapsulate single reading in a function

```
int salinity_power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int salinity_input_pin = 2; //  Analog input pin
  int salinity;

  salinity = salinity_reading( salinity_power_pin, salinity_input_pin );
  Serial.println(salinity);
}
// ---------------------------------------------------
int salinity_reading( int power_pin, int input_pin ) {

  int reading;

  digitalWrite( power_pin, HIGH );    //  Turn on the sensor
  delay(100);                         //  Wait to settle
  reading = analogRead( input_pin );  //  Read voltage
  digitalWrite( power_pin, LOW );     //  Turn off the sensor
  return reading;
}
```
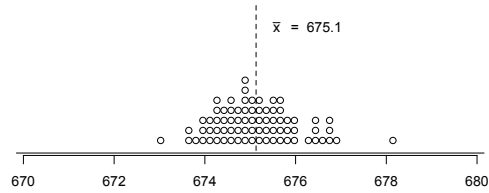
Value of the local variable called "reading" is returned and stored in the variable called "salinity.

Improve the function: Average several readings

Average is a measure of central tendency

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$\bar{x} = 675.1$

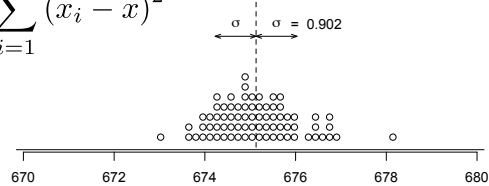670   672   674   676   678   680

Improve the function: Compute standard deviation

Standard deviation is a measure of spread

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i$$

$$\sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} \left(x_i - \bar{x}\right)^2}$$

σ $\sigma = 0.902$

670   672   674   676   678   680

First improvement: Average several readings

This loop() function does not need to know the details

```
int salinity_power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int salinity_input_pin = 2; //  Analog input pin
  int nave = 20;              //  Number of readings to average
  float salinity;            //  Float stores fractional reading from computed average

  salinity = salinity_reading_average( salinity_power_pin, salinity_input_pin, nave );
  Serial.println(salinity);
}
```

First improvement: Average several readings

This loop() function does not need to know the details

```
int salinity_power_pin = 4;   //  Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);
  pinMode(power_pin, OUTPUT);
}

void loop()
{
  int salinity_input_pin = 2; //  Analog input pin
  int nave = 20;              //  Number of readings to average
  float salinity;            //  Float stores fractional reading from computed average

  salinity = salinity_reading_average( salinity_power_pin, salinity_input_pin, nave );
  Serial.println(salinity);
}
```

`salinity_reading_average` is a user-written function to compute the average of nave samples from the salinity sensor

---

First improvement: Average several readings

Details are hidden in read_salinity_average

```
float salinity_reading_average( int power_pin, int input_pin, int nave ) {

  int   i;
  float reading, sum;  //  Use floats for more precision and to prevent overflow of sum

  sum = 0.0;
  for ( i=1; i<=nave; i++ ) {

    digitalWrite( power_pin, HIGH );  //  Supply power to the sensor
    delay(100);                       //  Wait for sensor to settle
    sum += analogRead( input_pin );   //  Add reading to the running sum
    digitalWrite( power_pin, LOW );   //  Turn off power to the sensor
    delay(10);                        //  wait between readings

  }

  reading = sum/float(nave);
  return reading;
}
```

Compute average and standard deviation

Code is more complex

- ❖ C functions can only "return" one value
- ❖ C functions can modify inputs that are passed by address
- ❖ The address of a variable is its location in memory
- ❖ The address can be assigned to another variable called a pointer
- ❖ Pointers are challenging for the beginner

A simple example of pointers

Pass the *value of* x into the function

Pass the *address of* y into the function

```
void loop() {

  int x = 2;
  int y;

  change_value(x, &y);
}


// ---------------------------------

void change_value( int a, int *b ) {

  *b = 2*a;
}
```

A simple example of pointers

Pass the *value of* x into the function

Pass the *address of* y into the function

```
void loop() {

  int x = 2;
  int y;

  change_value(x, &y);
}
```

*b is the *pointer to* (the address of) the second input argument

```
// -------------------------------

void change_value( int a, int *b ) {

  *b = 2*a;
}
```

change what is stored in *b

A simple example of pointers

Pass the *value of* x into the function

Pass the *address of* y into the function

```
void loop() {

  int x = 2;
  int y;

  change_value(x, &y);
}
```

*b is the *pointer to* (the address of) the second input argument

```
// -------------------------------

void change_value( int a, int *b ) {

  *b = 2*a;
}
```

change what is stored in *b

Note: `change_value` does not return a value — its return type is `void`. When `change_value(x,&y)` is executed, the value stored in `y` is changed.

Compute average and standard deviation

```
void salinity_reading_stats( int power_pin, int input_pin, int nave, float *ave, float *stdev ) {

  int    i, n;
  float  sum;                    //  Use a float to prevent overflow
  float  reading[BUFFER_LENGTH]; //  Array to store readings

  n = min( nave, BUFFER_LENGTH );  //  Make sure we don't over-run the data buffer

  // -- Store readings in an array
  for ( i=0; i<n; i++ ) {                // First array index is 0, last is n-1
    digitalWrite( power_pin, HIGH );     //  Supply power to the sensor
    delay(100);                          //  Wait for sensor to settle
    reading[i] = analogRead( input_pin ); //  Add reading to the running sum
    digitalWrite( power_pin, LOW );      //  Turn off power to the sensor
    delay(10);                           //  wait between readings
  }

  // -- Compute average and standard deviation.
  for ( sum=0.0, i=0; i<n; i++ )  {
    sum += reading[i];
  }
  *ave =  sum/float(nave);

  for ( sum=0.0, i=0; i<n; i++ ) {
    sum += pow(reading[i] - *ave, 2);
  }
  *stdev = sqrt( sum/float(n-1) );
}
```

Use salinity_reading_stats

```
int salinity_power_pin = 4;   //  Digital I/O pin

#define  BUFFER_LENGTH  100   //  Size of array to store readings for computation of ave and stdev
                             //  Reduce BUFFER_LENGTH to save memory if statistics are OK
                             //  with smaller sample size
void setup()
{
  Serial.begin(9600);
  pinMode(salinity_power_pin, OUTPUT);
}

void loop()
{
  int    salinity_input_pin = 2;
  int    nave = 20;
  float ave, stdev;

  salinity_reading_stats( salinity_power_pin, salinity_input_pin, nave, &ave, &stdev);

  Serial.print(ave);  Serial.print(", ");   Serial.println(stdev);
}
```

Pass the *address of* ave and *address of* stdev

Use ave and stdev as normal variables

Learning C++ Pointers for REAL Dummies
http://alumni.cs.ucr.edu/~pdiloren/C++_Pointers/