Arduino Programming
Part 5: Functions Redux and Intro to Arrays

ME 121

Mechanical and Materials Engineering
Portland State University
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 (optional)
- 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
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 $\infty$?

What is the input reading for each of those conditions?

If the resistance varied linearly with salinity, would the voltage vary linearly with salinity?
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
int salinity_power_pin = 4;  // Digital I/O pin, Global variable

void setup()
{
  Serial.begin(9600);  //Open communication with Serial Monitor
  pinMode (salinity_power_pin, OUTPUT); //set digital pin 4 to be an output pin
}

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

  digitalWrite (salinity_power_pin, HIGH);  // Provide power to sensor
  delay (100);  // Wait to settle (100ms)
  reading = analogRead (salinity_input_pin);  // Read voltage
  digitalWrite (salinity_power_pin, LOW);  // Turn off the power to sensor

  Serial.println (reading);  // show reading value in serial monitor
}
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

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

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

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

    salinity = sensor_reading(salinity_power_pin, salinity_input_pin);
    Serial.println(salinity);
}

// ---------------------------------------------
int sensor_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;
}
```
Use a function to make a single reading

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

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

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

    salinity = sensor_reading (salinity_power_pin, salinity_input_pin);
    Serial.println (salinity);
}

// -----------------------------------------------
int sensor_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

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

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

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

    salinity = sensor_reading (salinity_power_pin, salinity_input_pin);
    Serial.println (salinity);
}

int sensor_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 `sensor_reading`
Encapsulate single reading in a function

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

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

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

    salinity = sensor_reading(salinity_power_pin, salinity_input_pin);
    Serial.println(salinity);
}

// ---------------------------------------------
int sensor_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”.

- **Encapsulate single reading in a function**
- **int salinity_power_pin = 4; // Digital I/O pin, Global variable**
- **void setup()**
  ```cpp
  {
      Serial.begin(9600);
      pinMode(salinity_power_pin, OUTPUT);
  }
  ```
- **void loop()**
  ```cpp
  {
      int salinity_input_pin = 2; // Analog input pin
      int salinity;
      
      salinity = sensor_reading(salinity_power_pin, salinity_input_pin);
      Serial.println(salinity);
  }
  ```
- **// ---------------------------------------------**
- **int sensor_reading (int power_pin, int input_pin)**
  ```cpp
  {
      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”.
  ```cpp
  ```
```
Improve the function:
Average several readings

Average is a measure of central tendency

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

\[ \bar{x} = 675.1 \]
Improve the function: Compute standard deviation

Standard deviation is a measure of spread

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

\[ \sigma = \sqrt{\frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2} = 0.902 \]
```cpp
int salinity_power_pin = 4; // Digital I/O pin, Global variable

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

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

    salinity = sensor_reading(salinity_power_pin, salinity_input_pin);
    Serial.println(salinity);
}

// Function previously written (no average)
int sensor_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;
}
```
### Improvement: Average several readings

```cpp
int salinity_power_pin = 4; // Digital I/O pin, Global variable
void setup()
{
    Serial.begin(9600);
    pinMode(salinity_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 = sensor_reading_average(salinity_power_pin, salinity_input_pin, nave);
    Serial.println(salinity);
}

// -----------------------------------------------
float sensor_reading_average(int power_pin, int input_pin, int n)
{
    int i;
    float reading;
    float sum; // Use floats for more precision and to prevent overflow of sum
    sum = 0.0;
    for (i = 1; i <= n; i++)
    {
        digitalWrite(power_pin, HIGH); // Turn on the sensor
        delay(100); // Wait to settle
        sum += analogRead(input_pin); // Add reading to the running sum
        digitalWrite(power_pin, LOW); // Turn off the sensor
        delay(10); // Wait between readings
    }
    reading = sum / float(n);
    return reading;
}
```
**Improvement: Average several readings**

```c
int salinity_power_pin = 4;      // Digital I/O pin, Global variable
void setup()
{
  Serial.begin(9600);
  pinMode (salinity_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 = sensor_reading_average (salinity_power_pin, salinity_input_pin, nave);
  Serial.println (salinity);
}
// ---------------------------------------------------------------
float sensor_reading_average (int power_pin, int input_pin, int n){
  int i;
  float reading;
  float sum;               // Use floats for more precision and to prevent overflow of sum
  sum = 0.0;
  for (i=1 ; i<=n ; i++){
    digitalWrite (power_pin, HIGH);               // Turn on the sensor
    delay (100);                                   // Wait to settle
    sum += analogRead (input_pin);                // Add reading to the running sum
    digitalWrite (power_pin, LOW);                // Turn off the sensor
    delay(10);                                    // Wait between readings
  }
  reading = sum/float(n);
  return reading;
}
```
Introduction to Arrays and Pointers

Goal: Be able to have a function with more than one return value (Here, average and standard deviation)

- Optional -
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

```c
void loop() {
    int x = 2;
    int y;
    change_value(x, &y);
}

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

void change_value(int a, int *b) {
    *b = 2*a;
}
```

Pass the value of `x` into the function

Pass the address of `y` into the function
A simple example of pointers

```c
void loop() {
    int x = 2;
    int y;
    change_value(x, &y);
}

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

void change_value(int a, int *b) {
    *b = 2*a;
}
```

Pass the value of `x` into the function

Pass the address of `y` into the function

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

change what is stored in `*b`
A simple example of pointers

```c
void loop() {
    int x = 2;
    int y;
    change_value(x, &y);
}

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

void change_value(int a, int *b) {
    *b = 2*a;
}
```

Pass the value of \( x \) into the function

Pass the address of \( y \) into the function

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

Note: `change_value` does not return a value. Its return type is `void`. When is executed, the value stored in \( y \) is changed.

change what is stored in \*\( b \)
Compute average and standard deviation

```c
int salinity_power_pin = 4; // Digital I/O pin, Global variable
#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; // Analog input pin
  int nave=20; // Number of readings to average
  float salinity_ave, salinity_stdev;

  sensor_reading_stats (salinity_power_pin, salinity_input_pin, nave, &salinity_ave, &salinity_stdev);
  Serial.print (salinity_ave); Serial.print (", "); Serial.println (salinity_stdev);
}

void sensor_reading_stats (int power_pin, int input_pin, int n, float *ave, float *stdev)
{
  int i, k;
  float sum;
  float reading[BUFFER_LENGTH]; // Array to store readings
  k = min( n, BUFFER_LENGTH); // Make sure we don’t over-run the data buffer

  // Store readings in an array
  for (i=1 ; i<=k ; i++)
  {
    digitalWrite (power_pin, HIGH); // Turn on the sensor
    delay (100); // Wait to settle
    reading[i] = analogRead (input_pin); // Add the value of reading[i] to the array
    digitalWrite (power_pin, LOW);
    delay(10); // Turn off the sensor
    // Wait between readings
  }

  // Compute average and standard deviation
  for (sum=0.0, i=0; i<k ; i++)
  {
    sum += reading[i]; // Calculate sum of all readings to be used to calculate average
  }
  *ave = sum/float(n);
  for (sum=0.0, i=0; i<k ; i++)
  {
    dev += pow (reading[i]-*ave,2); // Calculate sum (using readings and average) to be used to compute stdev
  }
  *stdev = sqrt(sum/float(k-1) );
}
```