**Deadline (Preliminary draft): Feb 5, 2023 – 11:30 pm**

**Deadline (Final Draft): Feb 12, 2023 – 11:30 pm**

**Possible points:** 100 points

**Section 1. Measuring inputs**

In this studio, each team will implement a water tank control system. Using the sensors given in class, each team will use PEEB to manage water without any human intervention.

The sections below outline the deliverables' requirements and steps, including graphical and code elements.

**Parts you need to complete sections 1.1 and 1.2**

* PEEB kit
* USB Cable
* Water shield
* Wires – Use these to insert the water level sensor into the water.
* Water level sensor
* Magnet

**Getting familiar with the Water Shield**

The water shield (see Figure 1) belongs to the family of educational kits for Arduino that form a part of the PEEB project. This shield uses several sensors, LEDs, and internally connected buttons.

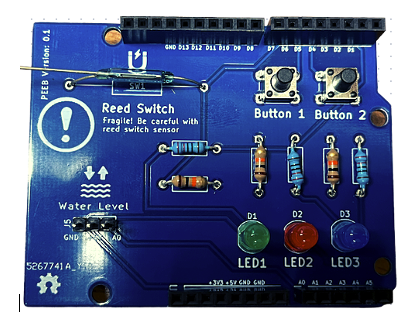


Figure 1. Water shield

Table 1 outlines the connection of each element with its respective pints.

Table 1. Internal Connections

|  |  |
| --- | --- |
| Components | Pin |
| Button 1 | D5 |
| Button 2 | D4 |
| Reed switch | D6 |
| Water level | A0 |
| LED1 | D1 |
| LED2 | D2 |
| LED3 | D3 |

**PEEB setup**

Before running the experiment today, make sure you complete the steps below:

**Step 1.** Attach the water shield to Arduino

**Step 2.** Plug in the USB cable

**Step 3.** Open the Arduino IDE

* 1. **Conditional statement**

Since computers cannot think on their own, and human beings want to put computers into performing complex and even simple works, software needs to be designed to run on the computer system and give it certain instructions to follow. Conditional statements, expressions, or simply conditionals are features of programming languages that tell the computer to execute certain actions, provided certain conditions are met.

In this first project, you will create your first Arduino program to control the behavior of one LED based on a button. To do so, use an **if** statement to change the output conditions based on changing the input conditions.

The **if()** statement is the most basic of all programming control structures. It allows you to make something happen or not, depending on whether a given condition is true or not. It looks like this:

if(someCondition){

//do stuff if the condition is true

}

There is a common variation called if-else that looks like this:

if(someCondition){

//do stuff if the condition is true

} else {

//do stuff if the condition is false

}

There's also the else-if, where you can check a second condition if the first is false:

if(someCondition){

//do stuff if the condition is true

} else if (anotherCondition) {

//do stuff only if the first condition is false

//and the second condition is true

}

You can use the if and else statements in Arduino to check multiple conditions by using logical operators such as && (and), || (or), and ! (not). For example, to check if two conditions are true:

if (condition1 && condition2) {

// code to execute if both conditions are true

} else {

// code to execute if either condition is false

}

To check if either one of two conditions are true:

if (condition1 || condition2) {

// code to execute if either condition is true

} else {

// code to execute if both conditions are false

}

You can also chain multiple if-else statements together to check multiple conditions:

if (condition1) {

// code to execute if condition1 is true

} else if (condition2) {

// code to execute if condition1 is false and condition2 is true

} else {

// code to execute if both conditions are false

}

**Conditional statement – Practice**

In this example, you assign a pin for Button 1 (Button 1 is already connected to D4 on Water shield) and set Button 1 as INPUT, and a pin for the LED 2 (LED 2 is already connected to **D2**) as set it as OUTPUT. You will tell the Arduino to turn the LED on as long as the button is being pressed and to keep the LED off when the button is not being pressed. When the button is released, the circuit breaks, and the LED will turn off again.

Different from the last class, where you declared each pin number manually, start your code by defining **constants**. By doing so, you do not need to change your pin number individually next time you make changes in your code. Constants allow you to uniquely name things in the program, but unlike variables, they cannot change. Name the input and output for easy reference, and create another named constant to hold the buttonState values equal to zero.

**Note:** *You must declare it before void setup.*

const int buttonPin = 4; //Pin connected to button 1

const int ledPin = 2; //Pin connected to LED 2

int buttonState = 0; //give pushbutton a value

The name **const** represents the constant keyword. It modifies the behavior of the variables in our program. It further makes the variable as 'read-only'. The variable will remain the same as other variables, but its value cannot be changed.

In the **setup( )**, you are going to set the LED pin as output and the button pin as input.

pinMode(ledPin,OUTPUT); //set LED pin as output

pinMode(buttonPin,INPUT); //set button pin as input

In the **loop( )**, you will use the variables defined earlier in your code. First, you read the button state by using the function **digitalRead** (this function reads the value from a specified digital pin, either HIGH or LOW).

buttonState = digitalRead(buttonPin);

In this section, you are going to use the statement **if**. More specifically, if the pushbutton is pressed, set the output as HIGH. If the pushbutton is not pressed, the output as LOW. In other words, you’ve told the Arduino to compare two things. When comparing two things in programming, you use two equal signs ==. If you use only one sign, you will be setting a value instead of comparing it.

if (buttonState == HIGH){

digitalWrite(ledPin,HIGH);

}

else{

digitalWrite(ledPin,LOW);

}

**Did you know it?** In Arduino, relational operators are used to compare two values and return a Boolean value (either true or false) based on the outcome of the comparison. The following relational operators are commonly used in Arduino programming:

* == (equal to)
* != (not equal to)
* > (greater than)
* < (less than)
* >= (greater than or equal to)
* <= (less than or equal to)

For example, the statement if (x == 5) will evaluate to true if the variable x is equal to 5, and false otherwise. Similarly, the statement if (x > y) will evaluate to true if the variable x is greater than the variable y, and false otherwise. These operators are used in conditions to control the flow of the program.

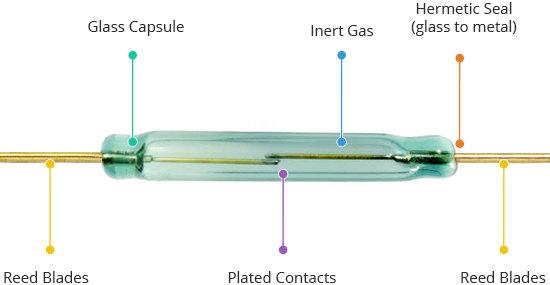
**Use it**

After writing your code, time to test your first conditional statement. First, verify your code by pressing **Verify** . Then, press UPLOAD . Once your Arduino is programmed, you should see the LED off. When you press the Button 1, the LED will turn on.

**NOTE:** *If you see any error message, read the error message and check for any missing element of your code, such as brackets or semi-colons.*

**Your turn!**

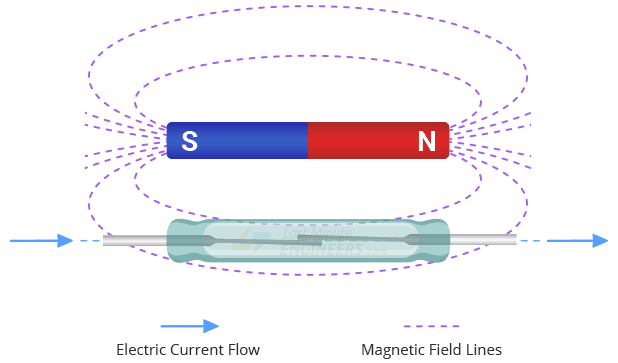
In this practice, you will familiarize yourself with a sensor called Reed Switch (see Figure 2). This sensor acts as a button that turns on or off when the magnet is nearby. Reed switches are used in various creative applications like door sensors, anemometers (determine the wind speed), etc. They are perfect for any project that requires non-contact control. So, let’s take a closer look at what they are and how they work!



**Figure 2.** Reed switch sensor

A typical reed switch contains a pair of metal reeds made of a ferromagnetic material. The reeds are hermetically sealed inside a tubular glass envelope to keep them free of dust and dirt. The hermetic sealing of reed switches makes them suitable for use in explosive environments where small sparks from conventional switches would constitute a hazard. The glass tube is filled with an inert gas, usually nitrogen, or a vacuum to prevent oxidation of the contacts.

The key to understanding how reed switches work is to realize that they are part of a magnetic circuit, as well as an electrical one – magnetism flows (See Figure 3) through them as well as electricity. As you bring a magnet closer to the reed switch, the entire switch becomes a part of the “magnetic circuit” including the magnet (the dotted line in the image shows part of the magnetic field).



**Figure 3.** Magnetic field acting on reed switch

The two contacts of a reed switch become opposite magnetic poles, which is why they attract and snap together. It doesn’t matter which end of the magnet you bring closer: the contacts still polarize in opposite ways and attract each other. A reed switch like this is normally open (NO). This means ‘normally’ when the switch is unaffected by the magnetic field, the switch is open and does not conduct electricity. When a magnet comes close enough to activate the switch, the contacts close and current flows through. In other words, the reed switch is equivalent to a push button that uses magnetic field instead of your finger

**Your task: Create one application where you turn a LED on once you activate a reed switch (PIN: D6) AND button 1 at the same time. Call your instructor once you are done.**

***NOTE:*** *In some boards, the reed switch is placed as normally open (NO). In other boards, it is placed is normally closed (NC). Normally open - Is a contact that does not flow current in its normal state. Energizing it and switching it on will close the contact, causing it to allow current flow. Normally closed - Is a contact that flows current in its normal state.*

* 1. **Analog signals: analog input, using the serial monitor**

Until now, our sketches have been using digital electrical signals with two discrete levels, HIGH and LOW. Unlike digital signals, analog signals can vary with indefinite numbers of step between high and low. It is possible to get information from analog sensors to measure things like temperature, light, and pressure. To do this, you will take advantage of the Arduino’s built-in Analog-to-Digital Converter (ADC). Analog in pins A0-A5 can be used to do so.

**Analog sinals – Practice**

You will be using a water level sensor to measure the water level using analog input. This sensor has ten exposed copper traces (see Figure 4), five of which are power traces, and the remaining five are sense traces. These traces are interlaced so that there is one sense trace between every two power traces. Normally, power and sense traces are not connected, but when immersed in water, they are bridged.

The water level sensor is extremely simple to use and only requires three pins to connect.

Diagram, schematic

Description automatically generated

Figure 4. Water level sensor (Pinout)

Update figure!

* **Analog Signal (S)** is an analog output pin that will be connected to one of your Analog inputs
* **+VCC (+)** pin provides power to the sensor. It is recommended that sensor be powered from 3.3V to 5V.
* **Ground (-)** = negative (GND)

The Arduino IDE comes with a tool called the serial monitor that enables you to report back results from the microcontroller. Using the serial monitor (see Figure 5), you can get information about the status of sensors and get an idea about what is happening in your circuits.

Graphical user interface, application

Description automatically generated

Figure 5. How to access the serial monitor

In this initial part, you must check the water level using the serial monitor. Just as you have been doing in earlier projects, wire up your circuit according to the pinout in Table 2.

|  |  |
| --- | --- |
| **Water shield connections** | **Water sensor connections** |
| A0 | S |
| GND | - |
| +5V | + |

Start your code by defining constants. Constants allow you to uniquely name things in the program, but unlike variables they cannot change. Name the analog input for easy reference, and create another named constant to hold the baseline water level. Different than section 1.1, you will learn another strategy to define pinout using #define. #define is like a placeholder. The Arduino compiler replaces all mentions of this constant with its value at the compile time. This means that the values defined using #define don't take up any program space. In general, the const keyword is preferred for defining constants and should be used instead of #define. But let’s use define in this example. Type the following code before void setup.

#define sensorPin A0

Next, create a variable to receive the information from the sensor. Set its value to zero.

int value = 0; // variable to store the sensor value

In the setup you are going to do two things: 1) set up pin mode and 2) initialize the serial port to the desired speed. You will use a new command, **serial.begin**( ). This opens up a connection between the Arduino IDE and the computer, so you can see the values from the analog input on your computer screen. The 9600 is the speed at which the Arduino will communicate, 9600 bits per second.

Serial.begin(9600);

In the loop ( ), you need to read the sensor output. You can use the function **analogRead** to do so. More specifically, you will assign the value from your sensor to the variable ***value*** by using **analogRead**( ). The value, which is between 0 and 700 is a representation of the voltage on the pin.

value=analogRead(sensorPin);

The function Serial.print( ) sends information from the Arduino to a connected computer. If you give Serial.print( ) an argument in quotation marks, it will print out the text you typed. If you give it a variable as an argument, it will print out the value of that variable.

  Serial.print("Sensor reading=");

  Serial.println(value);

  delay(500); // Check for new value every 5 sec

**Use it**

After writing your code, time to test your first conditional statement. First, verify your code by pressing **Verify** . Then, press UPLOAD . Once your Arduino is programmed, open the serial monitor and test your code. **Call your instructor** before testing your application.

**Your turn!**

In this practice, you will familiarize yourself with multiple tasks: conditional statements and analog reading.

**Your task:** *Create a code that active LED 3 on Water Shield once you have two conditions happening at the same time:*

* *Condition 1: Water level sensor detects a certain amount of water (you decide which output)*
* *Condition 2: Button 1 is pressed*

***Call your instructor once you are done.***

**Functions learned so far:**

* digitalRead
* digitalWrite
* analogRead
* serial.print
* delay

**Statements learned so far:**

* Conditional statement (if-else)

**Source:** https://www.arduino.cc/reference/en/

**Section 2. Assignment**

Assignment goal: Create a water tank control system to collect rainwater. You must use the functions and statements learned in the class so far. No physical prototype is needed to build the water tank in this assignment.

Project requirements:

* Your project must contain two sensors (water level and reed switch).
* Your project must have one or more outputs (e.g., light, sound, etc.) to indicate the different water levels inside the tank. You should have a minimum of four different states, from empty to full tank.

Resources available to be used as output (per team):

* 6x LEDs
* 6x resistors
* 1x speaker

**Deliverable 1 - Deadline (Preliminary draft): Feb 5, 2023 – 11:30 pm**

**Challenge 1.** **a)** Sketch a detailed water tank diagram, including all dimensions, sensor locations, and any additional information that will aid in understanding the proposed design and expected outputs. **Be creative!!** You can use hand drawing or any other software in this first draft. Also, include **b)** the expected programming logic behind your circuit and how you intend to implement that according to the project requirements described above. In this initial draft, **DO NOT** include any code. You must develop the code in the classroom. You can use flowcharts ([More information here](https://www.edrawsoft.com/explain-algorithm-flowchart.html)) or any other diagram.

**IMPORTANT:** Keep in mind that no substantial changes to the project and code logic will be allowed once the draft is submitted. Discuss a realistic plan with your team before the deadline. If you implement a solution different than what you proposed in your draft, you must provide a clear justification behind your changes and why you did change it. During next class, you **CANNOT** use any document besides your draft and the [Arduino documentation](https://www.arduino.cc/reference/en/) to develop your code.

**Deliverable 2 - Deadline (Final Memo): Feb 12, 2023 – 11:30 pm**

**Challenge 2.** The final memo **must** contain a professional representation of your design and steps to build and test your prototype. You must implement the project described in Challenge 1 and write a technical memo describing the process. For this project, the ***theory*** will include conditional statements and digital versus analog signals. You must do your research to include additional information about each of these theories. Do not forget to cite your references. Your ***method*** must include the planning process described in your draft and the implementation process to execute your plan. Your final memo must be written in a way that any person will be able to replicate your process.

**Grading Rubric**

|  |  |  |
| --- | --- | --- |
| Description | Expectation | Met? |
| Formatting – Final Memo (80 points) | | |
| General | All instructions and formatting standards were followed. | /2 |
| Writing is in past tense and third person. | /2 |
| Code included as text (**NOT** screenshot) – Code must be placed in Appendix | /3 |
| Figures, labels, and equations formatted correctly | /3 |
| Front cover properly formatted | /3 |
| Abstract is clear and complete | /3 |
| Contents include every single section in your document | /2 |
| Introduction | Clear and concise introduction to paper | /2 |
| Theory | Ties together information from class. Author’s introduction demonstrates a theoretical understanding of conditional statements | /7 |
| Ties together information from class. Author’s introduction demonstrates a practical understanding of digital and analog signals | /7 |
| Authors used at least one external reference for each theory | /5 |
| Methods | The paper communicates the methods well. Circuit schematics, tables, and equations provide important details to the reader and are properly labeled and described | /10 |
| The report clearly outlines the tools and procedures used for project design | /5 |
| Discussion | Robust discussion of your results and decisions for each challenge | /10 |
| Conclusion | Meaningful conclusion, and includes ideas not already discussed in paper | /10 |
| References | References properly organized and cited | /3 |
| Appendix | Appendix properly organized and cited | /3 |
| Challenges (20 points) | | |
| Challenge 1  (Submit as a simple memo) | Project design includes all dimensions, sensors, and expected water level output | /4 |
| Students demonstrated systematic thinking behind the water tank | /2 |
| Clear code planning | /2 |
| Organization is clear and professional | /2 |
| Challenge 2  (Submit as a final memo) | Students followed the initial plan presented in Challenge 1 | /3 |
| Project is functional | /4 |
| Project followed the requirements | /3 |
| Total score = | | /100 |