

Tutorial 05 – Security and Intrusion Detection

Make sure you complete [Setup and Test Your Adafruit Feather Huzzah32 Connection](#) document before starting this Tutorial 05.

Learning Objectives

1. Understand the use of the magnetic contact switch, capacitive touch sensor pins, and piezo buzzer.
2. Understand the difference between Advanced Encryption Standard (AES) and Secure Hash Algorithms (SHA).
3. Understanding the basics of programming, i.e., `char`, `unsigned char`, string, pointers (*) and array ([]), formatted string using a format specifier, and the differences between a constant pointer and a pointer to constant.
4. Implement built-in functions from `mbedtls/aes.h` library, i.e., `mbedtls_aes_init()`, `mbedtls_aes_setkey_enc()`, `mbedtls_aes_setkey_dec()`, `mbedtls_aes_crypt_ecb()`, `mbedtls_aes_free()` in the code in order to perform symmetric encryption via Adafruit Feather Huzzah32.
5. Implement built-in functions from `mbedtls/md.h` library, i.e., `mbedtls_md_init()`, `mbedtls_md_setup()`, `mbedtls_md_starts()`, `mbedtls_md_update()`, `mbedtls_md_finish()`, `mbedtls_md_free()` in the code in order to hash data via Adafruit Feather Huzzah32.
6. Implement function calls from Espressif's LED Control (LEDC) Application Programming Interface (API), i.e., `ledcSetup()`, `ledcAttachPin()`, `ledcWriteTone()`, `ledcWrite()` to generate different sound frequencies and volume through the use of Pulse Width Modulation (PWM) on Adafruit Feather Huzzah32.
7. Implement function calls related to string, i.e., `sizeof()`, `strlen()`, `sprintf()`, and getting input from touch pins, i.e., `touchRead()` on Adafruit Feather Huzzah32.

Theory

An array is a collection of data (of the same data type in C programming language) that allows random access, i.e., by using an index number. For example, `myArray[8]` will directly access the array element with index 8 as shown in Figure 1 without having to go through all the elements prior to that chosen element sequentially.

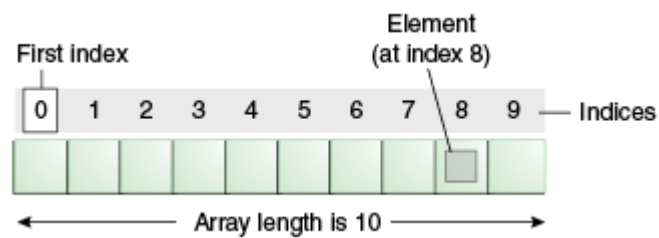


Figure 1: A 10 elements array example (Oracle, 2022).

As shown in Figure 1, arrays are zero indexed, i.e., the first element of the array is at index 0. The last element of the array is indexed as $(\text{length} - 1)$ or 9 in this example. Figure 2 illustrates an example of array declaration and one of many possible ways to initialise an array, i.e., using an initialiser list.

```
int myArray[10]={9, 3, 2, 4, 3, 2, 7, 8, 9, 11};
// myArray[9]    contains 11
// myArray[10]   is invalid and contains random information (other memory address)
```

Figure 2: An array example (Arduino, 2022a).

Based on Figure 2, `myArray` has a length or a size of 10, the first element of `myArray` or `myArray[0]` has a value of 9, and the last element of `myArray` or `myArray[9]` has a value of 11. The curly braces `{ }` are one of the commonly used ways for the initialisation of arrays, especially when multiple values are to be initialised. The initialiser list initialises elements of the array in the order of the list. Like any other data types in the Arduino language, it is also possible to initialise (or assign a new value to) each element of the array individually using an assignment operator (`=`). For example, `myArray[3] = 12;` will replace the value 4 with 12 in Figure 2 example.

Accessing the memory location past the end of the array, i.e., `myArray[10]`, is probably not going to cause much problems except for getting some invalid data. However, writing to a memory location past the end of the array can cause the program to crash or malfunction. As the Arduino language is based on the C programming language, accessing a memory location past the end of the array (i.e., array out of bound) will not stop the compilation of the sketches (i.e., no compilation error).

Read the following Arduino reference pages to better understand array.

1. Array: <https://www.arduino.cc/reference/en/language/variables/data-types/array/> .

Data types for numerical data and boolean data, e.g., `float`, `double`, `int`, `bool`, have been looked at in previous tutorials. In the Arduino language, a variable with `char` data type is used to store a character value, encoded as a number using an ASCII chart (Figure 3). For example, a variable storing character 'A' would store the number 65 in its memory location. It should be remembered that when assigning a character to a variable, a pair of single quotes, i.e., ' ' surrounding that character is required. Another key point to note is that ' ' is not the same as " ". The latter is common in MS Word or when copying and pasting code from other sources. The wrong pair of single quotes will generate an error during compilation of sketches.

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL	{null}	32	20	040	Space	64	40	100	64;	@	96	60	140	96;	`
1	1	001	SOH	{start of heading}	33	21	041	!	65	41	101	65;	A	97	61	141	97;	a
2	2	002	STX	{start of text}	34	22	042	"	66	42	102	66;	B	98	62	142	98;	b
3	3	003	ETX	{end of text}	35	23	043	#	67	43	103	67;	C	99	63	143	99;	c
4	4	004	EOT	{end of transmission}	36	24	044	\$	68	44	104	68;	D	100	64	144	100;	d
5	5	005	ENQ	{enquiry}	37	25	045	%	69	45	105	69;	E	101	65	145	101;	e
6	6	006	ACK	{acknowledge}	38	26	046	&	70	46	106	70;	F	102	66	146	102;	f
7	7	007	BEL	{bell}	39	27	047	'	71	47	107	71;	G	103	67	147	103;	g
8	8	010	BS	{backspace}	40	28	050	(72	48	110	72;	H	104	68	150	104;	h
9	9	011	TAB	{horizontal tab}	41	29	051)	73	49	111	73;	I	105	69	151	105;	i
10	A	012	LF	{NL line feed, new line}	42	2A	052	*	74	4A	112	74;	J	106	6A	152	106;	j
11	B	013	VT	{vertical tab}	43	2B	053	+	75	4B	113	75;	K	107	6B	153	107;	k
12	C	014	FF	{NP form feed, new page}	44	2C	054	,	76	4C	114	76;	L	108	6C	154	108;	l
13	D	015	CR	{carriage return}	45	2D	055	-	77	4D	115	77;	M	109	6D	155	109;	m
14	E	016	SO	{shift out}	46	2E	056	.	78	4E	116	78;	N	110	6E	156	110;	n
15	F	017	SI	{shift in}	47	2F	057	/	79	4F	117	79;	O	111	6F	157	111;	o
16	10	020	DLE	{data link escape}	48	30	060	0	80	50	120	80;	P	112	70	160	112;	p
17	11	021	DC1	{device control 1}	49	31	061	1	81	51	121	81;	Q	113	71	161	113;	q
18	12	022	DC2	{device control 2}	50	32	062	2	82	52	122	82;	R	114	72	162	114;	r
19	13	023	DC3	{device control 3}	51	33	063	3	83	53	123	83;	S	115	73	163	115;	s
20	14	024	DC4	{device control 4}	52	34	064	4	84	54	124	84;	T	116	74	164	116;	t
21	15	025	NAK	{negative acknowledge}	53	35	065	5	85	55	125	85;	U	117	75	165	117;	u
22	16	026	SYN	{synchronous idle}	54	36	066	6	86	56	126	86;	V	118	76	166	118;	v
23	17	027	ETB	{end of trans. block}	55	37	067	7	87	57	127	87;	W	119	77	167	119;	w
24	18	030	CAN	{cancel}	56	38	070	8	88	58	130	88;	X	120	78	170	120;	x
25	19	031	EM	{end of medium}	57	39	071	9	89	59	131	89;	Y	121	79	171	121;	y
26	1A	032	SUB	{substitute}	58	3A	072	:	90	5A	132	90;	Z	122	7A	172	122;	z
27	1B	033	ESC	{escape}	59	3B	073	;	91	5B	133	91;	[123	7B	173	123;	{
28	1C	034	FS	{file separator}	60	3C	074	<	92	5C	134	92;	\	124	7C	174	124;	
29	1D	035	GS	{group separator}	61	3D	075	=	93	5D	135	93;]	125	7D	175	125;	}
30	1E	036	RS	{record separator}	62	3E	076	>	94	5E	136	94;	^	126	7E	176	126;	~
31	1F	037	US	{unit separator}	63	3F	077	?	95	5F	137	95;	_	127	7F	177	127;	DEL

Source: www.LookupTables.com

Figure 3: ASCII chart (ASCII Table, n.d.).

A word or a sentence, AKA a string, is usually stored as an array of `char`. An array of `char` can be initialised individually using an assignment operator (i.e., `=`), initialiser list, or as a string literal. Although the Arduino language has a `String` object, it is outside the scope for 5COM2004, and it uses more memory than an array of `char` or a string literal. Figure 4 illustrates examples of declaring and initialising arrays of `char` through initialiser lists (i.e., `Str2`, `Str3`) containing individual characters, and through string literal (i.e., `Str4`, `Str5`, `Str6`). In the Arduino language, a

string is denoted using a pair of double quotes, e.g., "arduino". Similar to the case of char, " " is not the same as " " and the latter will cause a compilation error.

```
char Str1[15];
char Str2[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o'};
char Str3[8] = {'a', 'r', 'd', 'u', 'i', 'n', 'o', '\0'};
char Str4[] = "arduino";
char Str5[8] = "arduino";
char Str6[15] = "arduino";
```

Figure 4: Array of char examples (Arduino, 2022c).

It should be noted that a string literal gets a `Null` character, i.e., `'\0'`, automatically added at the end to terminate the string, and therefore sufficient memory spaces should be allocated, or data can be overwritten. In Figure 4, you can note that "arduino" is only 7 characters, but at least 8 memory spaces are allocated. For `Str4` in Figure 4, 8 is automatically allocated as the size of that array. It should be noted that whilst `sizeof()` returns the actual size of the array, i.e., 8, `strlen()` excludes the Null character and will return 7.

The `Null` character is important for the printing of strings, e.g., using `print()` or `println()`, as these functions ONLY stop reading/printing at `Null` character. Based on example shown in Figure 4, we can assume `Str3` is located in a memory location right after `Str2`. If your sketch contains `Serial.println(Str2);`, the sketch will print `arduinoarduino`, because of the lack of `Null` character at the end of `Str2`, `println()` carries on reading/printing whatever data that follows the memory location after `'o'`. If your program is behaving strangely, check that all your strings end with a `Null` character.

Figure 5 illustrates the differences between `char` and `unsigned char`. Instead of having 1 bit to store the sign and 7 bits to store the character, `unsigned char` has 8 bits in total to store the character, and therefore can represent a range of value between 0 to 255 instead of 0 to 127 with a plus or a minus sign.

Read the following reference pages to understand 3 different data types for storing characters and utilities functions which are introduced in this tutorial.

1. `char`: <https://www.arduino.cc/reference/en/language/variables/data-types/char/> .
2. `unsigned char`: <https://www.arduino.cc/reference/en/language/variables/data-types/unsignedchar/> .
3. `string`: <https://www.arduino.cc/reference/en/language/variables/data-types/string/> .

4. `sizeof()`: <https://www.arduino.cc/reference/it/language/variables/utilities/sizeof/> .
5. `strlen()`: <https://man7.org/linux/man-pages/man3/strlen.3.html> .

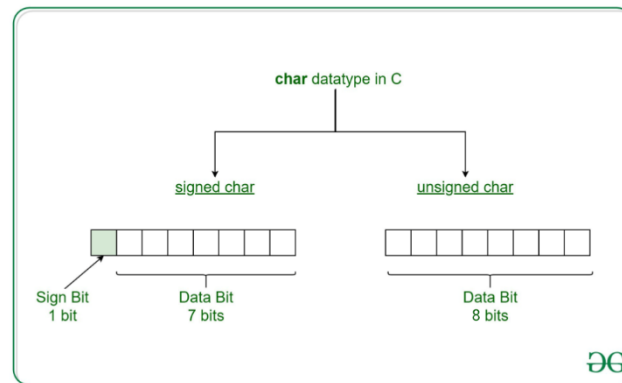


Figure 5: Signed and unsigned char (Geeksforgeeks, 2020).

Every variable is a memory location, and every memory location has an address to it. In the example in Figure 6, when you declare a variable, e.g., `b`, a memory location is allocated for that variable. For `b` in this example, this location has an address of 1008. When you assign a value to `b`, e.g., `b = 2;`, then 2 will be stored at 1008 (although this memory location is currently empty in Figure 6 example). If the variable is an array, then a consecutive block of memory addresses large enough to fit all elements of that array will be allocated. For example, if we declare `char b[2];`, then two consecutive memory addresses (e.g., 1008 and 1009) will be allocated and will belong to the variable `b`. Figure 6 illustrates the relationship between a pointer, a variable and their memory addresses.

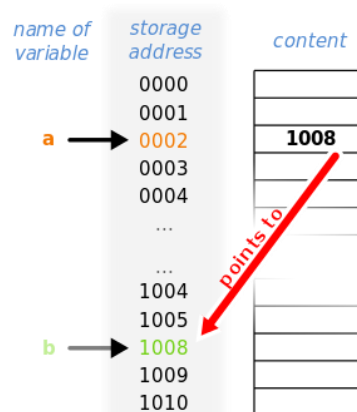


Figure 6: Visual example of a pointer, a variable, and memory addresses.

Most data types store a value, e.g., from file, from computation, or from user input, in their memory location. A pointer is a special data type, which unlike any other data types, stores a memory

address. In Figure 6, the variable `a` is an example of a pointer. Instead of storing a value, variable `a` (which has a memory address of 0002) stores the memory address of `b`, i.e., 1008. This relationship is indicated using the red arrow in Figure 6 and it can also be read as variable `a` is a pointer pointing to `b`. It is important to note that the data type of the pointer must match the data type of the variable it is pointing to, i.e., an `int` pointer is needed to point to the memory address of an `int` variable.

Like the C programming language, there are 2 pointer access operators in the Arduino language, i.e., dereference (`*`) and reference (`&`). Whilst dereferencing allows the access of the **value of the variable the pointer variable is pointing to**, referencing allows the access of the **memory address of that variable**. Figure 7 illustrates the use of the 2 pointer access operators.

```
int *p;      // declare a pointer to an int data type
int i = 5;
int result = 0;
p = &i;      // now 'p' contains the address of 'i'
result = *p; // 'result' gets the value at the address pointed by 'p'
             // i.e., it gets the value of 'i' which is 5
```

Figure 7: Referencing and dereferencing using pointer access operators (Arduino, 2022b).

Based on Figure 7, `p` stores the memory address of `i` through referencing, i.e., `&i`, and `result` is given the value of `i` through dereferencing, i.e., `*p`. If `i` was allocated a memory address 1111, then `&i` would produce 1111. In Figure 7 example, `*p` would produce 5 because variable `p` is a pointer pointing to variable `i`.

Read the following Arduino reference pages to understand pointer access operators, which are introduced in this tutorial.

2. Dereferencing (`*`): <https://www.arduino.cc/reference/en/language/structure/pointer-access-operators/dereference/>.
3. Referencing (`&`): <https://www.arduino.cc/reference/en/language/structure/pointer-access-operators/reference/>.

There are many uses for pointers, but for 5COM2004, the focus will only be on the passing of variables, specifically an array, to a function. You have written your own functions in Tutorial 03. Please revisit Practice 3.2 if you don't remember how to write functions. Figure 8 (Left) illustrates an example of a function from Practice 3.2.

```

int addition(int x, int y)           int addition(int *x, int *y)
{
    return (x + y);                 {
}                                   return (*x + *y);

```

Figure 8: (Left) Pass by value function, (Right) Pass by reference.

When parameters are passed to a function by value, the program copies the values of these parameters into new memory locations. The `x` and `y` in Figure 8 (Left) are examples of passed by value parameters. The variable `x` and `y` inside `addition()` have a local scope and only exist until the end of the function call. If changes are made to `x` and `y` inside `addition()` (Left), these changes will NOT affect the values of the variables where `x` and `y` are copied from.

Alternatively, parameters can be passed to a function by reference using pointer access operators. In Figure 8 (Right), the `x` and `y` parameters receive the references of the variables when and from where the `addition()` is called. If changes are made to `x` and `y` inside `addition()` (Right), these changes will affect the values of the variables where `x` and `y` are referenced to. Figure 9 illustrates the differences between pass by reference and pass by value.

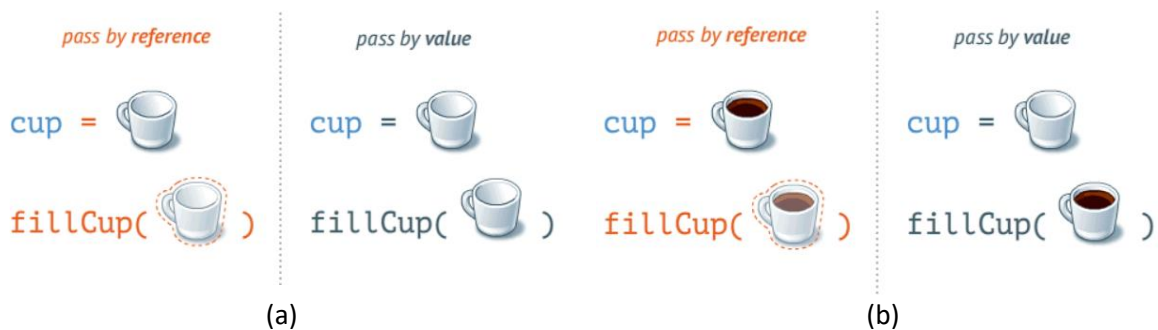


Figure 9: Coffee filling – pass by reference vs. pass by value (Penjee.com, 2022).

Figure 9 (a) represents the outcome once the functions are called. The coffee cup in Figure 9 represents `x` or `y` in Figure 8. In pass by reference, a link (or reference, i.e., a pointer) to the coffee cup outside of `fillCup()` is created (from where the call to `fillCup()` was initiated). In pass by value, a copy of the coffee cup (as denoted by a coffee cup without the orange dotted surrounding line) is created and the value (i.e., empty) of that coffee cup is copied over (from where the call to `fillCup()` was initiated), WITHOUT any link to the cup outside of `fillCup()`.

Figure 9 (b) represents the outcome during the execution of the function. In pass by reference, because of the link (or the reference), the changes can be seen in both of the cups (i.e., inside and outside of `fillCup()`). In pass by value, because there is no link (or reference), the changes to the coffee cup inside `fillCup()` cannot be seen in the cup outside of `fillCup()`.

After the functions exit, both the pointer (from pass by reference) and the copy of the coffee cup (from pass by value) are destroyed. For the case of pass by value, this means that the changes to the copy of the coffee cup inside `fillCup()` will be lost. However, for the case of pass by reference as the changes has already been reflected in the cup outside of `fillCup()` during the execution of the function through the use of pointer, these changes are not lost when the function exits.

Like other data types, pointers themselves can be a constant (i.e., a constant pointer). A constant pointer must be declared and initialised at the same time, e.g., `int const *ptr1 = &a;`. This type of pointer forbids the change of memory location the pointer is pointing to through referencing (&). A statement such as `ptr1 = &b;` following that declaration statement will cause a compilation error because the statement attempts to change the memory location the pointer `ptr1` is pointing to, i.e., to be changed from the memory location of variable `a` to the memory location of variable `b`. A statement such as `*ptr1 = 2;`, however, will NOT cause compilation error.

There is also a (non-constant) pointer to constant. This type of pointer forbids changes made to the value of the variable the pointer is pointing to through dereferencing (*). However, unlike a constant pointer, a (non-constant) pointer to a constant does not need to be declared and initialised at the same time, e.g., `const int *ptr2; ptr2 = &a;`. The pointer `ptr2` can be assigned to point at different variables or memory locations, i.e., a statement such as `ptr2 = &b;` will NOT cause a compilation error. However, a statement such as `*ptr2 = 2;` will cause compilation error, as the value of the variable `a` in this example cannot be changed through the pointer.

There is also a constant pointer to constant. This type of pointers forbids the changes made to the value of the variable the pointer is pointing to through dereferencing (*) and it also forbids the change of memory location the pointer is pointing to through referencing (&). Similar to a constant pointer, a constant pointer to constant also needs to be declared and initialised at the same time, i.e., `const int const *ptr3 = &a;`. Statements such as `ptr3 = &b;` or `*ptr3 = 2;` will cause compilation errors because `ptr3` can neither be changed to point to variable `b` instead of variable `a`, nor can `ptr3` be used to change the value of variable `a`.

Pointers are one of the most complicated topics in C programming. It is possible to write Arduino sketches without ever using pointers. However, knowledge of manipulating pointers can be useful in handling data structures such as arrays particularly for function calls. Unlike passing simple variables to functions, which are passed by value by default, arrays are by default passed by reference to functions. Performance benefits for passing by reference include speed and memory consumption, especially when the array is big.

The function `Serial.print()` and `Serial.println()` have been used in previous tutorials to print data to the Serial Monitor. These functions also have an optional parameter which specifies the number base (i.e., BIN, OCT, DEC, HEX for base 2, 8, 10, and 16 respectively) to be applied to the value of the first parameter if it is a whole number. For floating point numbers, this optional

parameter specifies the number of decimal places to use. Figure 7 shows examples of usage with respective output.

```
Serial.print(78, BIN) gives "1001110"  
Serial.print(78, OCT) gives "116"  
Serial.print(78, DEC) gives "78"  
Serial.print(78, HEX) gives "4E"  
Serial.print(1.23456, 0) gives "1"  
Serial.print(1.23456, 2) gives "1.23"  
Serial.print(1.23456, 4) gives "1.2346"
```

Figure 7: Formatting in `Serial.print()` (Arduino, 2022d).

Read the following Arduino reference pages, if you have already forgotten how to use `Serial.print()` and `Serial.println()`.

1. `Serial.print()`: <https://www.arduino.cc/reference/en/language/functions/communication/serial/print/> .
2. `Serial.println()`: <https://www.arduino.cc/reference/en/language/functions/communication/serial/println/> .

Instead of writing output directly onto the Serial Monitor, it is also possible to write the output into a data structure such as array. The `sprintf(destination, format_specifier, source)` allows an output from `source` to be formatted using a format specifier and stored in `destination`. It should be noted that the data structure for `destination` should be large enough to store the output as a terminating `Null` character is appended automatically to the content.

Read the following reference pages to understand `sprintf()` and the available format specifier in C programming language.

1. `sprintf()`: <https://man7.org/linux/man-pages/man3/sprintf.3p.html> .
2. Format specifier: <https://man7.org/linux/man-pages/man3/fprintf.3p.html> or output examples <https://cplusplus.com/reference/cstdio/printf/> .

-Magnetic door switches (Figure 8) are widely used in home security systems. One half of the switch set (the reed switch) should be set on a window, or a door frame and the other half (magnet) attached to the window or the door itself. When the switch set is separated from each other, the contact is broken and can be used to trigger an alarm.



Figure 8: Magnetic door switch (Proto-pic, 2023).

Just like switches or buttons, you do not need to differentiate the two wires of the magnetic door switch. When one wire is connected to Ground, and the other to an input pin with a pull-up resistor: the input pin is LOW when the magnet is near to the reed switch, and HIGH when the magnet is far from the reed switch.

Piezo buzzers (Figure 9) can make sounds, beeps, or play music. There are two pins on a piezo buzzer, but you do not need to differentiate them. One pin should be connected to a signal pin and the other pin to Ground.



Figure 9: Piezo buzzer (ESP32 I/O, n.d.)

Reference

Oracle. (2022). Arrays. [Website] Available at:

<https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html>

Arduino. (2022a). array. [Website] Available at:

<https://www.arduino.cc/reference/en/language/variables/data-types/array/>

Arduino. (2022b). *. [Website] Available at:

<https://www.arduino.cc/reference/en/language/structure/pointer-access-operators/dereference/>

ASCII Table. (n.d.). ASCII Table. [Website] Available at: <https://www.asciitable.com/>

Arduino. (2022c). string. [Website] Available at:

<https://www.arduino.cc/reference/en/language/variables/data-types/string/>

Geeksforgeeks. (2020). unsigned char in C with Examples. [Website] Available at:

<https://www.geeksforgeeks.org/unsigned-char-in-c-with-examples/#:~:text=unsigned%20char%20is%20a%20character,ranges%20from%200%20to%20255.>

Penjee.com. (2022). Passing by Value vs. by Reference Visual Explanation. [Website] Available at:

<https://blog.penjee.com/passing-by-value-vs-by-reference-java-graphical/>

Arduino. (2022d). Serial.print(). [Website] Available at:

<https://www.arduino.cc/reference/en/language/functions/communication/serial/print/>

Proto-pic. (2023). Magnetic Door Switch Set (Complete Assembly). [Website] Available at:

<https://proto-pic.co.uk/product/magnetic-door-switch-set-complete-assembly>

ESP32 I/O. (n.d.). ESP32 – Piezo Buzzer. [Website] Available at: <https://esp32io.com/tutorials/esp32-piezo-buzzer>

Practice 5.1

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial as well as the link to Espressif's mbedtls/aes.h below to make sure that you thoroughly understand the meaning of every line of the code below.

Hint: Espressif's mbedtls/aes.h - focus specifically the functions which are used in `encrypt()` and `decrypt()`

<https://github.com/espressif/Arduino-esp32/blob/39fb8c30440a4abd5fe0e2c87609ba6798ae8013/tools/sdk/include/mbedtls/mbedtls/aes.h> .

```
#include "mbedtls/aes.h"

void encrypt(const char *plainText, const char *key, unsigned char *outputBuffer)
{
    mbedtls_aes_context aes;

    mbedtls_aes_init(&aes);
    mbedtls_aes_setkey_enc(&aes, (const unsigned char*) key, (strlen(key) * 8));
    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_ENCRYPT, (const unsigned char*) plainText,
        outputBuffer);
    mbedtls_aes_free(&aes);
}

void decrypt(unsigned char *cipherText, const char *key, unsigned char *outputBuffer)
{
    mbedtls_aes_context aes;

    mbedtls_aes_init(&aes);
    mbedtls_aes_setkey_dec(&aes, (const unsigned char*) key, (strlen(key) * 8));
    mbedtls_aes_crypt_ecb(&aes, MBEDTLS_AES_DECRYPT, (const unsigned char*) cipherText,
        outputBuffer);
    mbedtls_aes_free(&aes);
}

void setup()
{
    Serial.begin(9600);
}

void loop()
{
    const char key[17] = "abcdefghijklmnop";
    const char plainText[17] = "1234567890123456";
```

```

unsigned char cipherTextOutput[17] = {'\0'};
unsigned char decipheredTextOutput[17] = "Not deciphered !";

encrypt(plainText, key, cipherTextOutput);

Serial.print("Original plain text: ");
Serial.println(plainText);

Serial.print("Ciphered text:");
for (int i = 0; i < 16; i++)
{
    char str[3];

    sprintf(str, "%02x", (int)cipherTextOutput[i]);
    Serial.print(str);
}
Serial.println();

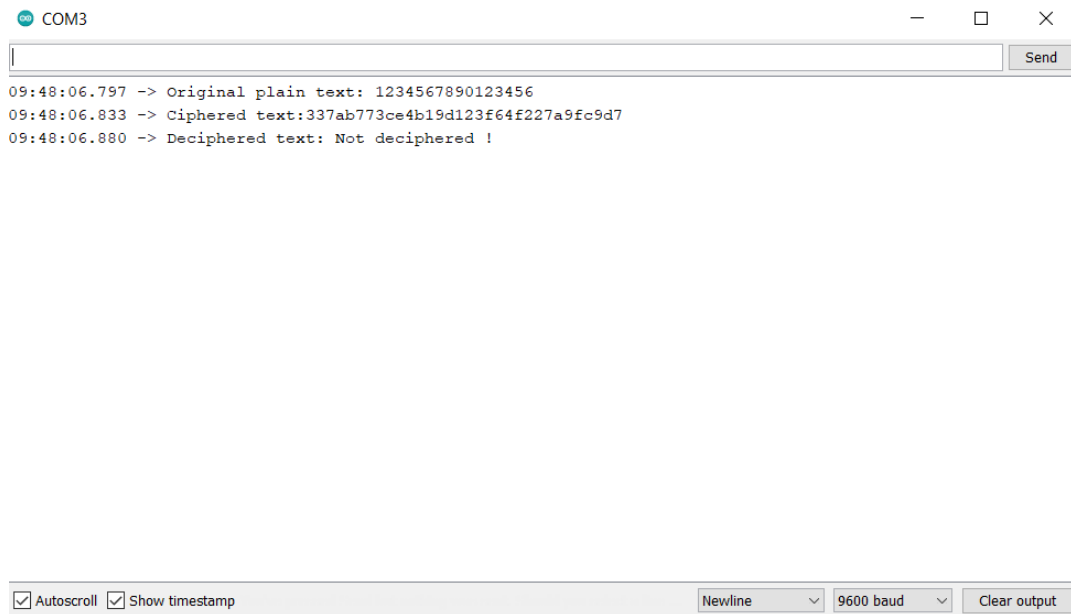
Serial.print("Deciphered text: ");
for (int i = 0; i < 16; i++)
{
    Serial.print((char)decipheredTextOutput[i]);
}
Serial.println();

delay(15000);
}

```

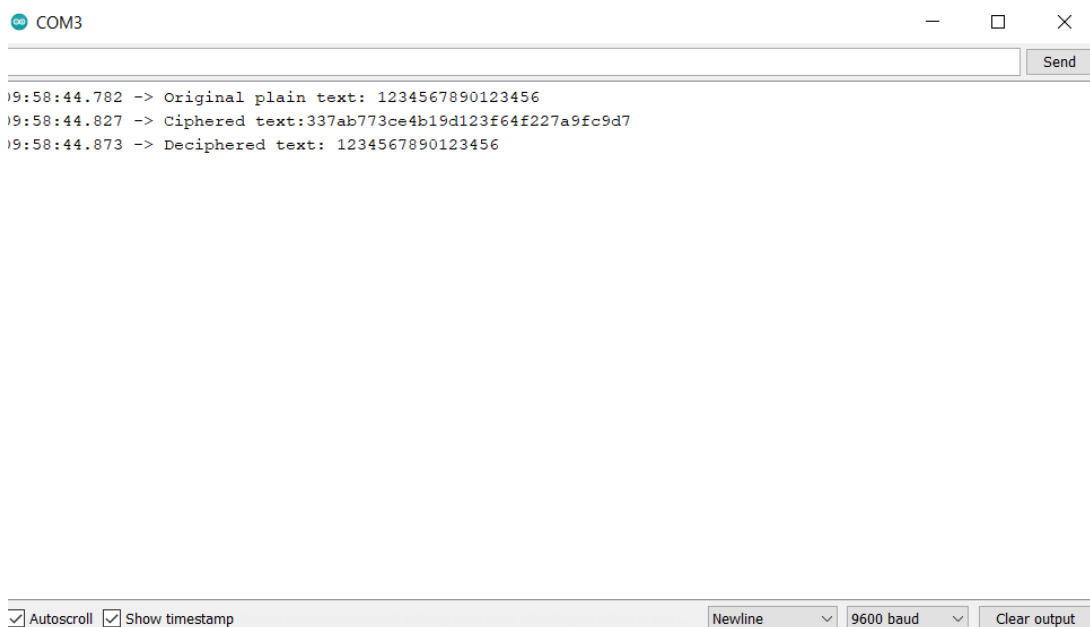
Note: Use default.json for WOKWI.

2. If you have a Serial Monitor open already, please close it.
3. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_05_aes. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
4. **Upload** the code.
5. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below.



TO DO

1. Modify tutorial_05_aes.ino to decrypt the `cipherTextOutput` and store the output in `decipheredTextOutput`.
2. If you have a Serial Monitor open already, please close it.
3. **Verify** your modification and **Upload** the code.
4. Go to **Tools** and then select **Serial Monitor**. You should now see the output of similar to what is shown below.



Practice 5.2

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial as well as the link to Espressif's mbedtls/md.h below to make sure that you thoroughly understand the meaning of every line of the code below.

Hint: Espressif's mbedtls/md.h - focus specifically the functions which are used in `hash()`
<https://github.com/espressif/arduino-esp32/blob/39fb8c30440a4abd5fe0e2c87609ba6798ae8013/tools/sdk/include/mbedtls/mdtls/md.h> .

```
#include "mbedtls/md.h"

void hash(const char *input, const size_t inputLength, unsigned char *outputBuffer)
{
    mbedtls_md_context_t ctx;
    mbedtls_md_type_t md_type = MBEDTLS_MD_SHA256;

    mbedtls_md_init(&ctx);
    mbedtls_md_setup(&ctx, mbedtls_md_info_from_type(md_type), 0);
    mbedtls_md_starts(&ctx);
    mbedtls_md_update(&ctx, (const unsigned char *) input, inputLength);
    mbedtls_md_finish(&ctx, outputBuffer);
    mbedtls_md_free(&ctx);
}

void setup()
{
    // put your setup code here, to run once:
    Serial.begin(9600);
}

void loop()
{
    // put your main code here, to run repeatedly:
    char in[32] = "SHA256SHA256SHA256SHA256SHA256!";
    byte out[32];

    delay(15000);
}
```

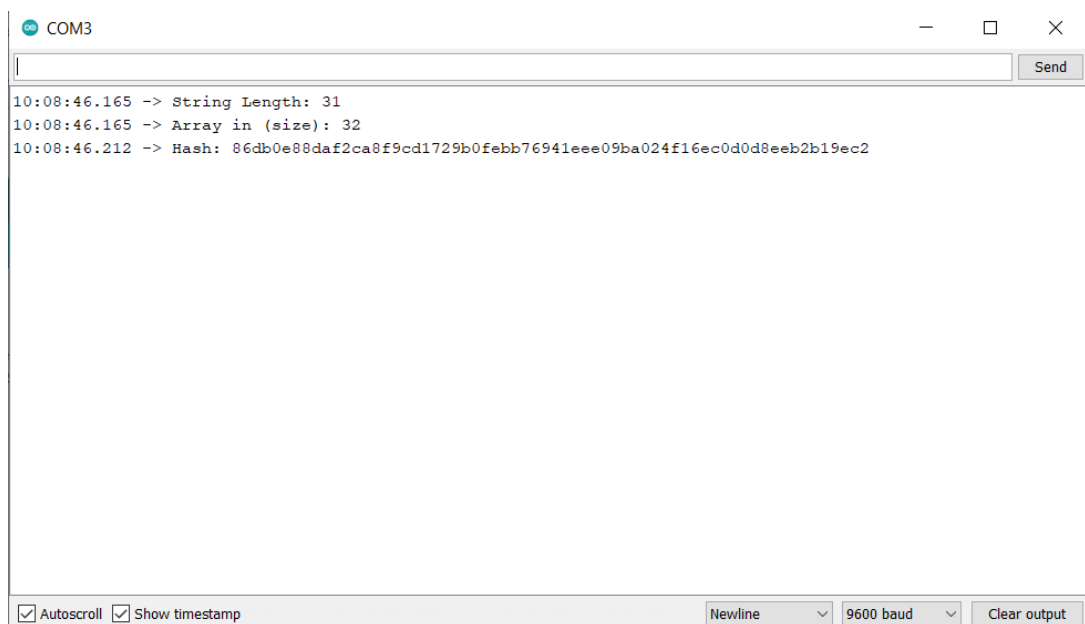
Note: Use default.json for WOKWI.

2. If you have a Serial Monitor open already, please close it.

3. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_05_sha. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
4. **Upload** the code.

TO DO

1. Modify tutorial_05_sha.ino to hash `in`, store the output in `out` and produce the output on the Serial Monitor similar to what is shown below.



The screenshot shows the Serial Monitor window for COM3. The output text is as follows:

```
10:08:46.165 -> String Length: 31
10:08:46.165 -> Array in (size): 32
10:08:46.212 -> Hash: 86db0e88daf2ca8f9cd1729b0febb76941eee09ba024f16ec0d0d8eeb2b19ec2
```

At the bottom of the window, there are checkboxes for 'Autoscroll' and 'Show timestamp', both of which are checked. To the right of these are dropdown menus for 'Newline' and '9600 baud', and a 'Clear output' button.

Note: Revisit the Theory part if you do not understand why the length of the string is different from the size of the array.

2. If you have a Serial Monitor open already, please close it.
3. **Verify** your modification and **Upload** the code to see if it still works correctly.

Practice 5.3

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
int LEDPin = 15;
int doorPin = 21;

void setup() {
  // put your setup code here, to run once:
  Serial.begin(9600);
  pinMode(doorPin, INPUT_PULLUP);
  pinMode(LEDPin, OUTPUT);
}

void loop() {
  // put your main code here, to run repeatedly:

}
```

Note: Revisit Tutorial 04, if you have forgotten what INPUT_PULLUP does.

2. Connect magnetic contact switch, LED diode and resistor based on the information given in the sketch above.

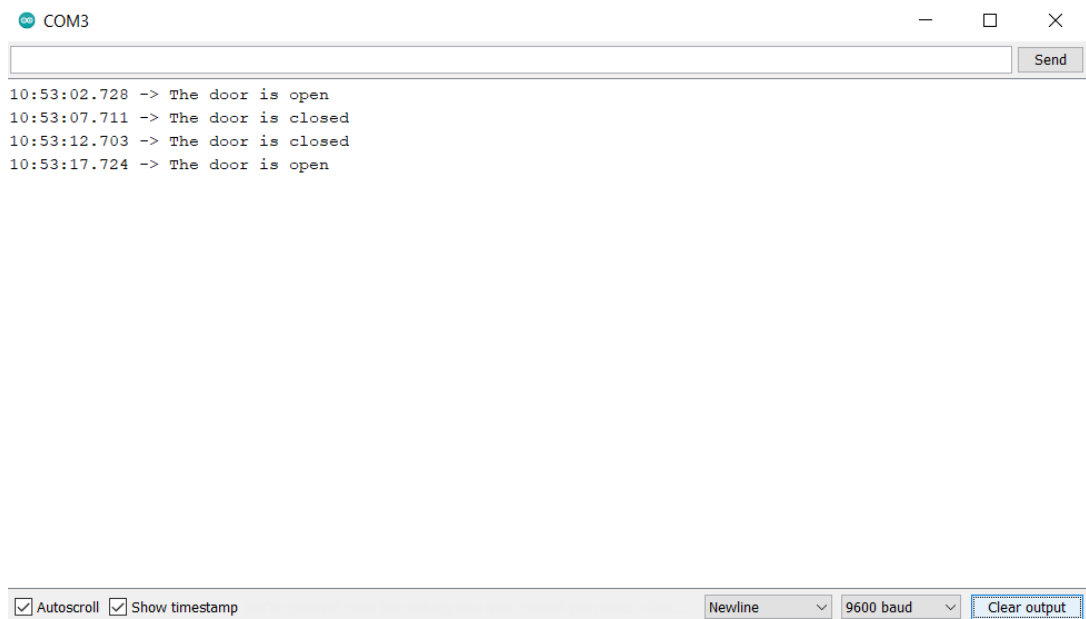
Note: Revisit Lecture 05, if you are not sure how the magnetic contact switch works.

3. If you have a Serial Monitor open already, please close it.
4. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_05_door_LED. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
5. **Upload** the code.

Note: Use practice5_3.json for WOKWI. There is no door sensor on WOKWI, so a button is used instead (i.e., press and hold to simulate closing the door).

TO DO

1. Modify tutorial_05_door_LED.ino to produce the output on the Serial Monitor similar to what is shown below, i.e., when the magnetic does not contact the reed switch, produce the message "The door is open" and light up the LED diode, and when the magnetic contacts the reed switch, produce the message "The door is closed" and turn off the LED diode.



2. If you have a Serial Monitor open already, please close it. **Verify** your modification.
3. **Upload** the code.
4. Go to **Tools** and then select **Serial Monitor** to see if it still works correctly.

Practice 5.4

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
const int buzzerPin = 21;
const int toneChannel = 0;
const int toneFrequency = 800;

void setup()
{
  // put your setup code here, to run once:

  // setting up buzzer
  ledcSetup(toneChannel, toneFrequency, 8);
  ledcAttachPin(buzzerPin, toneChannel);

  Serial.begin(9600);
}

void loop()
{
  // put your main code here, to run repeatedly:

  ledcWriteTone(toneChannel, toneFrequency);

  // adjust duty cycle = volume of the buzzer
  ledcWrite(toneChannel, 10);
  delay(1000);
  ledcWrite(toneChannel, 100);
  delay(1000);
  ledcWrite(toneChannel, 255);
  delay(1000);

  ledcWrite(toneChannel, 100);
  // adjust frequency
  ledcWriteTone(toneChannel, 255);
  delay(1000);
  ledcWriteTone(toneChannel, 5000);
  delay(1000);
  ledcWriteTone(toneChannel, 10000);
  delay(1000);
}
```

Note: Use practice5_4.json for WOKWI.

2. Connect piezo buzzer based on the information given in the sketch above.

Note: Revisit Lecture 05, if you are not sure how the piezo buzzer works.

3. **Verify** the code. You will then be asked to save the sketch. Enter a **meaningful name** for your file, e.g., tutorial_05_buzzer. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
4. **Upload** the code.

TO DO

1. Modify the sketch so that it increases the volume from 0 up to 255 by 50 points per iteration. Afterward, it increases the frequency from 255 up to 10000 by 250 points per iteration.

Hint: Use loops.

2. **Verify** your modification and **Upload** the code to see if it works correctly.

Challenge 5.1

1. Type the code below into the coding area of your Arduino IDE. If you haven't done so already, use the links provided in the Theory part of this tutorial to make sure that you thoroughly understand the meaning of every line of the code below.

```
const int touchPin = T6; // GPIO pin 14
const int buzzerPin = 21;
const int toneChannel = 0;
const int toneFrequency = 800;
const int touchThreshold = 50; // turn on/off light if touchVal value < or > this threshold

void setup()
{
  // put your setup code here, to run once:
  Serial.begin(9600);

  // setting up buzzer
  ledcSetup(toneChannel, toneFrequency, 10);
  ledcAttachPin(buzzerPin, toneChannel);
}

void loop()
{
  // put your main code here, to run repeatedly:
  int touchVal; // TO DO: get touch value here

  ledcWriteTone(toneChannel, toneFrequency);

  Serial.print("Touch: ");
  Serial.println(touchVal);

  // TO DO: play sounds when you touch the sensor

  delay(500);
}
```

Note: Use challenge5_1.json for WOKWI. There is no simple touch pad on WOKWI, so a slider bar is used instead (i.e., instead of reading touch value this needs to be changed to analogRead()).

2. Connect the piezo buzzer and a jumper wire (for the capacitive touch sensor pin) based on the information given in the sketch above.

Note: Revisit Lecture 05, if you are not sure how the capacitive touch sensor pin works.

3. If you have a Serial Monitor open already, please close it.

4. **Verify** the code. You will then be asked to save the sketch. Enter a meaningful name for your file, e.g., tutorial_05_touch_challenge. Please **take note where you save your file**, so you know where to find it, should you need it afterwards.
5. **Upload** the code.

TO DO

1. Complete all of the TO DO tasks as shown in the comments in tutorial_05_touch_challenge.ino. You may need to adjust `touchThreshold` to an appropriate value.
2. If you have a Serial Monitor open already, please close it. **Verify** your modification.
3. **Upload** the code to see if it now produces the required output.