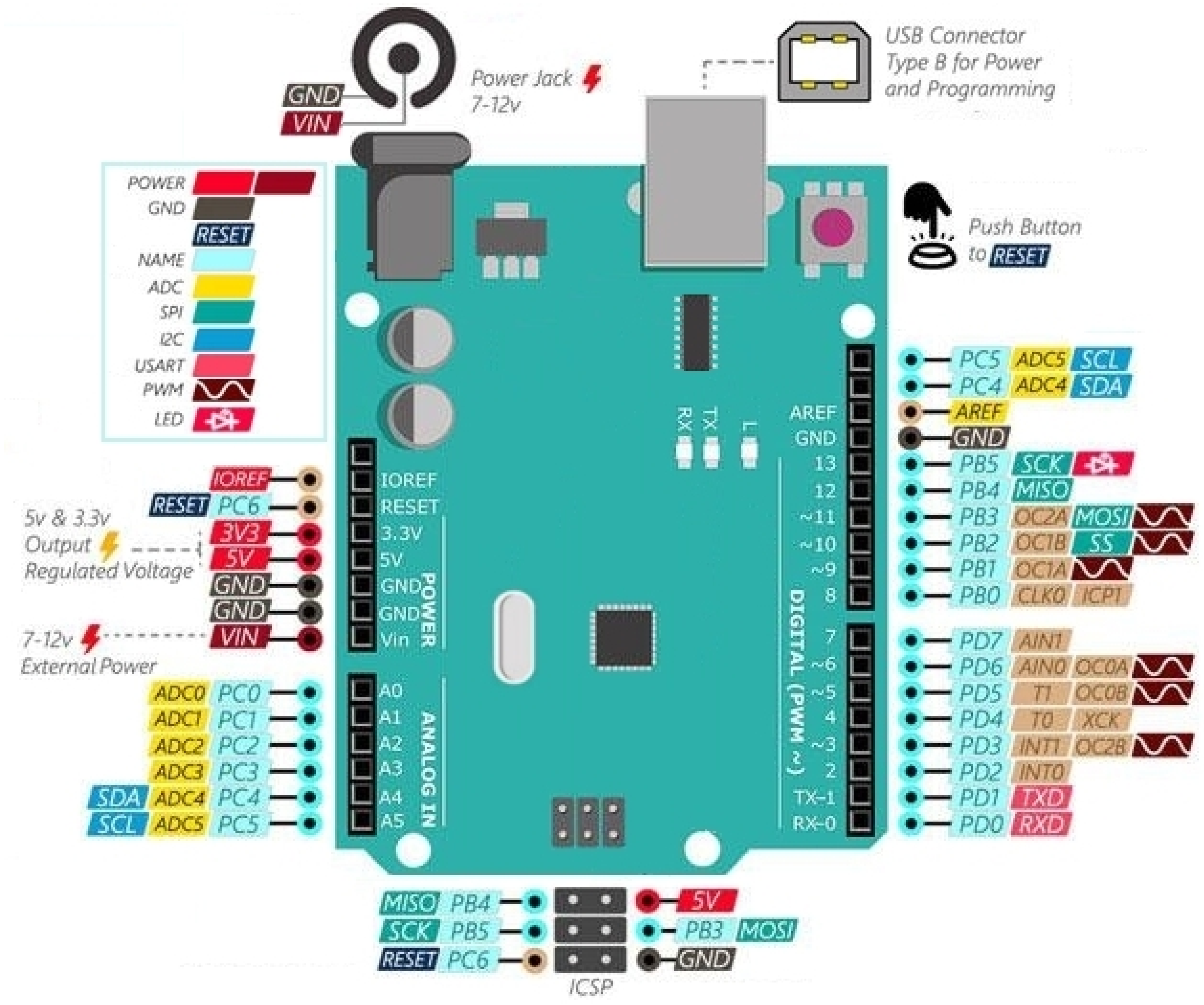
Arduino

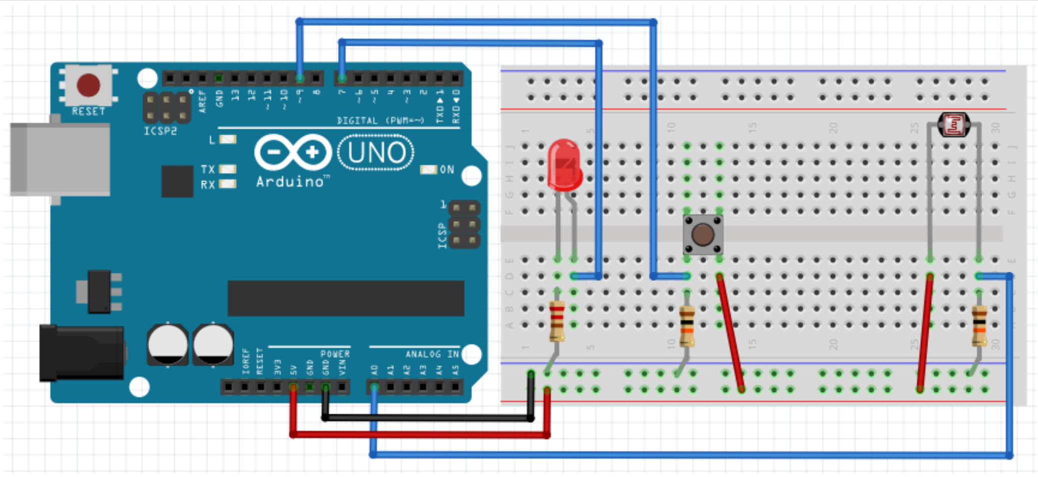
Tutorial 3
Arduino C-Language Programming

2021 by TeakSoon Ding
for STEMKRAF
https://github.com/teaksoon/stemkraf

Reference to the Arduino Uno board main components and its connections to the Atmega328 micro-controller



Parts Required for our C-Language Programming with Arduino
1 x Arduino UNO with USB 2.0 type A/B Data Cable connected to Computer with Arduino IDE Software installed
1 x Half-size Solderless Breadboard - Conductor
7 x Jumper Wire (Male to Male) - Conductor
1 x LED (Light Emitting Diode 5mm) - "raw" Output Device
1 x Resistor (220 Ohm) - Current Reducing Conductor
1 x Tactile Switch - "raw" Input Device
1 x Resistor (10K Ohm) - Current Reducing Conductor
1 x LDR (Light Dependent Resistor) "raw" Input Device (Sensor)
1 x Resistor (10K ohm) - Current Reducing Conductor



Keywords

Memory/ DataTypes	Program Control
01.void	21.return
02.const	22.if
03.static	23.else
04.signed	24.for
05.unsigned	25.while
06.short	26.do
07.char	27.switch
08.int	28.case
09.long	29.break
10.float	30.default
11.double	31.continue
12.sizeof	32.goto
13.struct	
14.union	
15.enum	
16.typedef	
17.auto	
18.register	
19.extern	
20.volatile	

Symbols

Program Control	Relation Logic	Math Ops.	Bit Ops.
#	==	*	|
< >	!=	%	&
“	<	/	^
‘	>	+	~
( )	<=	-	<<
{ }	>=		>>
[ ]			
//	**Logic**		
/* */	&&		
;	||		
,	!		
:			
?			
.			
\			
=			
Memory Address			
&			
*			

Below are not part of the C-Language but often used with the C-Language is the C-Language pre-processor.

The C-Language pre-processor give special instructions to the C-Compiler and also acts like a programable “Search and Replace” tool before the actual compilation starts.

#define	Similar to “Search and Replace”
#include	Insert extra codes from “.h” file
#undef	Undefines a #define
#ifdef	Returns true if #define exist
#ifndef	Returns true if #define does not exist
#if	Condition is true.
#else	Condition is false
#elif	Combination of #else and #if
#endif	Ends conditional
#error	Output error message
#pragma	Special Command to the C-Compiler

The Keywords and Symbols listed here cannot be used as Function Names, Variable Names, and any other types of Names in our C-Language Program.

C-Language only have 32 keywords and a bunch of Symbols. Out of that, many will not even be used in most of our Programs. This is a very simple and yet a very powerful Programming Language.

Almost all modern computer Operating System is written in C-Language. Most Runtime Engines for other Programming Language are also written in C-Language. When comes to micro-controllers, C-Language is also used almost everywhere.

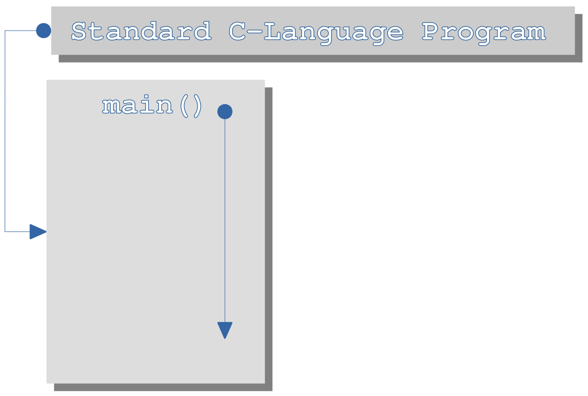
Standard C-Language Program

We must create at least **one function**,

- main() function

C-Language Program when executed by the CPU, will look for main() function and execute the C-Language Instruction Codes from there.

We need to put our C-Language Instruction Codes inside the main() function.



Arduino C-Language Program

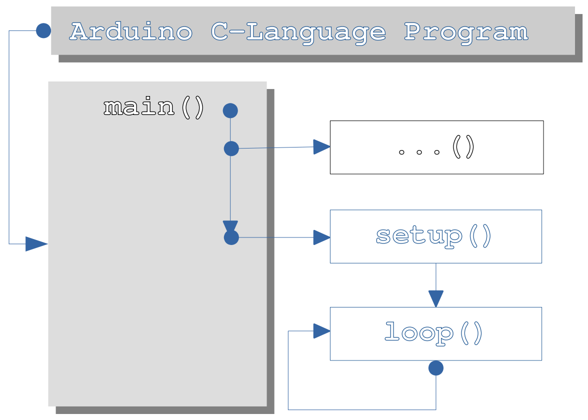
we must create at least **two functions**,

- setup() function
- loop() function

Arduino C-Language Program is also C-Language Program. Arduino C-Language Program when executed by the CPU, will also look for main() function and execute the C-Language Instruction Codes from there.

We not need to code the main() function with Arduino C-Language Program because the **main() function** along with many other functions(Arduino Library) **has already been coded for us.** They are just being **"hidden"** from us.

However, we need to code two functions in our Arduino C-Language Program, setup() and loop() function because the "hidden" main() function will Call that two functions.



Use the Arduino IDE Software
1.Save as "bare minimum arduino"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
void setup(){}
void loop(){}
```

We have no output to see in this Program

The setup() and loop() has to be coded by us in our Arduino C-Language Program because the "hidden" main() function expects them to exist

Below is a "simplified" version of the **"hidden" main() function**, the full version can be found in our Arduino IDE Software installation,
<installation folder>/hardware/arduino/avr/cores/arduino/main.cpp

```
// some codes here...
int main(void) {
  // some other codes...
  setup();
  for (;;) {
    loop();
  }
  return 0;
}
```

This time we change the “bare-minimum” Program a little bit, with some extra text in it

Use the Arduino IDE Software
1.Save as “bare minimum notes”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
/*
program name = bare minimum notes
Version = 1.01
*/
void setup(){} // do nothing
void loop(){} // do nothing
```

This program has no output to see (Notice that when compiled Program size is still the same vs the earlier Program, despite the additional text)

“/*” and “*/” symbol pair is for multiple line notes/comments. **Notes/comments** are Keyed-in within the symbol pair

```
/*
Program name = bare minimum
Version = 1.01
*/
```

“//” symbol is for single line notes/comments. **Notes/comments** are Keyed-in after the symbol

```
// do nothing
```

The C-Language Source Code in our Arduino IDE Software has to be converted into “Machine Code” (Compiled) before it can go into the Atmega328 micro-controller Program Memory (Flash Memory).

The notes/comments will not be “Compiled”, they will not be converted into “Machine Code”. They will NOT go into our micro-controller.

The notes/comments placed there for programmers reference in the Source Codes only. We can have as many notes/comments as we like because they will not effect our final Program that is stored in the micro-controllers Program Memory and run by the CPU.

We know there are things like **main()**, **setup()** and **loop()** from our "bare minimum" Program. All 3 are **functions**

What is a function ?

A C-Language function is an executable unit, consist of zero or more C-Language Instruction Codes.

We can do two things with function,

Define New function
= Key-in the content of the New function

Call Existing function
= Execute the content of an Existing function

A C-Language Program must have at least one function.

The main() function

Whenever the CPU executes a C-Language Program, the C-Language Program will automatically call the **main()** function.

The **main()** function is the starting point of a C-Language Program.

We can code our entire Program with a single main() function. However, very often we break them up multiple functions to make our Program easier to work with.

So, a bare minimum Program, we need to,
Define one New Function, The main() function (which will automatically be called when the Program starts)

After that is done, we can optionally, **Define more New Functions or Call Existing Functions**

The Arduino has actually caused a little confusion to those new to C-Language Programming.

Some may be wondering why we code the **setup()** and **loop()** functions instead of the **main()** function. First, they hide the main() function. Then, to make it worse, the Arduino IDE Software calls it a "Sketch".

So, Lets make it clear

This is a normal C-Language with a "hidden" main() function coded for us and used with a bunch of C-Language functions, collectively known as the "Arduino Library".

Define a New function

There are 4 Parts that needs to be coded

<ReturnDataType> <functionName>(<Parameters>){<InstructionCodes>}

STEP 1: Key-in Skeleton Code

```
void functionName(){}
```

STEP 2: Does the function have a return value ?

IF NO, Leave the “void” there
IF YES, Key-in DataType of returning value in <ReturnDataType> to replace “void”

STEP 3: Does the function requires value from calling function,<Parameters> ?

IF NO, Just leave the bracket pair “()” empty
IF YES, Within the bracket pair “()“, Key-in <DataType><space><MemoryName>. If more than one incoming values, Key-in multiple <DataType><space><MemoryName>, each separated with a comma “,”

STEP 4: Does the function have any C-Language instruction codes ?

IF NO, Just leave the curly bracket pair “{}” empty
IF YES, Within the curly bracket pair “{}“, Key-in C-Language instruction codes
IF “STEP 2:” have a return value, We must have the following additional C-Language instruction codes, within the curly bracket pair “{}”,

```
return <value>;
```

DataType of <value> must match the “**STEP 2:**” <ReturnDataType>

Example: Define setup() function

STEP 1: Key-in Skeleton Code
```
void setup(){}
```

STEP 2: Does the function have a return value ?
NO, void setup(){}

STEP 3: Does the function requires value from calling function,<Parameters> ?
NO, void setup(){}

STEP 4: Does the function have any C-Language instruction codes ?
NO, void setup(){}

Example: Define loop() function

STEP 1: Key-in Skeleton Code
```
void loop(){}
```

STEP 2: Does the function have a return value ?
NO, void loop(){}

STEP 3: Does the function requires value from calling function,<Parameters> ?
NO, void loop(){}

STEP 4: Does the function have any C-Language instruction codes ?
NO, void loop(){}

```
Example: Define setup() function with C-Language Instruction Codes
STEP 1: Key-in Skeleton Code
void setup(){}
STEP 2: Does the function have a return value ?
NO, void setup(){}
STEP 3: Does the function requires value from calling function,<Parameters> ?
NO, void setup(){}
STEP 4: Does the function have any C-Language instruction codes ?
YES,
void setup(){
  pinMode(9,OUTPUT);
  digitalWrite(9,HIGH);
}
```

Use the Arduino IDE Software (use hardware setup from pg36, for LED)
1.Save as “turn on LED”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = turn on LED
void setup(){
  pinMode(9,OUTPUT);
  digitalWrite(9,HIGH);
}
void loop(){}
```

Watch the LED connected to Arduino Uno connector “9”

```
Example: Define myBlink() function, uses all 4 parts of a function definition
STEP 1: Key-in Skeleton Code
void myBlink(){}
STEP 2: Does the function have a return value ?
YES, int myBlink(){}
STEP 3: Does the function requires value from calling function,<Parameters> ?
YES, int myBlink(int blinkTimes, unsigned long delayDuration){}
STEP 4: Does the function have any C-Language instruction codes ?
YES,
int myBlink(int blinkRepeat, unsigned long delayDuration){
  for (int i=0; i<blinkRepeat;i++) {
    digitalWrite(9,HIGH);delay(delayDuration);
    digitalWrite(9,LOW);delay(delayDuration);
  }
  return blinkRepeat;
}
```

Use the Arduino IDE Software
1.Save as “blink function”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = blink function
void setup(){
int blinkCount = 5;
  pinMode(9,OUTPUT);
  blinkCount = myBlink(blinkCount,500);
  blinkCount = blinkCount+5;
  blinkCount = myBlink(blinkCount,500);
}
void loop(){}

int myBlink(int blinkRepeat, unsigned long delayDuration) {
  for (int i=0; i<blinkRepeat;i++) {
    digitalWrite(9,HIGH);delay(delayDuration);
    digitalWrite(9,LOW);delay(delayDuration);
  }
  return blinkRepeat;
}
```

Watch the LED connected to Arduino Uno connector “9”

Call an Existing function.
Function is called based on the Function Definition (Header Part)

```
<ReturnDataType> <functionName>(<Parameters>){<InstructionCodes>}
```

STEP 1: Key in Skeleton Code

```
functionName();
```

STEP 2: Is the <ReturnDataType> entered as "void" ?

IF YES, Do Nothing, code is still functionName();
IF NO, Needs a Memory Storage with same DataType as <ReturnDataType> to receive the value from called function's "return <value>;"

```
<ReturnDataType> memoryName;

memoryName = functionName();
```

STEP 3: Does the function have Parameters ?
IF NO, Just leave the bracket pair "()" empty
IF YES, Within the bracket pair "()", Key-in <value> matching the Parameter.
If multiple Parameters, Key-in multiple <value> separated with comma ",", matching the Parameters in the Function Definition Header.

```
<ReturnDataType> memoryName;

memoryName = functionName(<value>,<value>,...);
```

Example: Call the **pinMode()** function

Function Definition Header for pinMode() function (from Arduino Library)

```
void pinMode(uint8_t pin, uint8_t mode);
```

STEP 1: Key in Skeleton Code

```
pinMode();
```

STEP 2: Is the <ReturnDataType> entered as "void" ?
YES, pinMode();

STEP 3: Does the function have Parameters ?
YES, pinMode(9,OUTPUT);

The pinMode() Function Defination Header can be found in Arduino.h, This file can be found in our Arduino IDE Software installation,

<installation folder>/hardware/arduino/avr/cores/arduino/Arduino.h

All the other Arduino Library function Defination can be found either in that file or somewhere in that same folder.

```
Example: Call the myBlink() function.

Function Definition Header for myBlink() function ( our own function )
int myBlink(int blinkRepeat, unsigned long delayDuration)

STEP 1: Key in Skeleton Code
myBlink();

STEP 2: Is the <ReturnDataType> entered as “void” ?
NO,
int blinkCount;

blinkCount = myBlink();

STEP 3: Does the function have Parameters ?
YES,
int blinkCount;
blinkCount = myBlink(5, 500);
```

This is the same “blink function” Program from previous example

```
// program name = blink function
void setup(){
int blinkCount;
  pinMode(9,OUTPUT);
  blinkCount = myBlink(5, 500);
}
void loop(){}

int myBlink(int blinkRepeat, unsigned long delayDuration) {
  for (int i=0; i<blinkRepeat;i++) {
    digitalWrite(9,HIGH);delay(delayDuration);
    digitalWrite(9,LOW);delay(delayDuration);
  }
  return blinkRepeat;
}
```

Watch the LED connected to Arduino Uno connector “9”

The setup() function contains C-Language Instruction Codes make “Calls” to functions
1. “pinMode() function” (function from Arduino Library)
2. “myBlink() function” (function coded by us)

C-Language Notes:

What is the following code for ?
`int blinkCount;`
We prepare a Memory Storage of “int” DataType, name it as blinkCount

What is the following code for ?
`=`
“=” is an assignment symbol, anything on the right will be assigned to the Memory Storage on the left of that symbol

What is the following code for ?
`;`
“;” signifies the end of a C-Language instruction code

This is the classic blink program that everyone will code when they were first introduced to Arduino Uno. We will use it to learn about code execution flow inside in an Arduino C-Language Program.

Use the Arduino IDE Software (use hardware setup from pg36, for LED)
1.Save as “classic blink”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = classic blink
void setup(){
  pinMode(9,OUTPUT);
}
void loop(){
  digitalWrite(9,HIGH);
  delay(500);
  digitalWrite(9,LOW);
  delay(500);
}
```

Watch the LED connected to Arduino Uno connector “9”

To understand our program flow, we must include the main() function (we use a simplified version in this example). Although we did not code the main) function in our Arduino IDE Software, it is included in our final “Machine Code” that will be running in the Atmega328 micro-controller.

```
01 int main(void){
02   setup();
03   for (;;) {
04     loop();
05   }
06   return 0;
07 }
```

```
08 // program Name = classic blink
09 void setup(){
10   pinMode(9,OUTPUT);
11 }
12 void loop(){
13   digitalWrite(9,HIGH);
14   delay(500);
15   digitalWrite(9,LOW);
16   delay(500);
17 }
```

Although we did not key in the item 01 to 07 in our Arduino IDE Software, it is included in our final “Machine Code” running in our micro-controller. **08 will not be in our “Machine Code”** because that is a “note/comment” line.

This is how our Program Flow, starting from left (from main() function),

01, 02, 09, 10, 11, 03, 04, 12, 13, 14, 15, 16, 17, 03, 04, 12, 13, 14...

03, 04, 12, 13, 14, 15, 16, 17, will be repeated because of the “for (;;)” instruction code inside the the main() function that calls the loop() function repeatadly.

The instuction code **for(;;){ }** will repeat everything inside the curly bracket pair indefinitely.

pinMode(), digitalWrite() and delay() are functions from the Arduino Library, which will run codes somewhere else but will return to the spot where it is called.

The pre-processor is not part of the C-Language but often used together with the C-Language. The pre-processor is like a programable "Search and Replace" tool in our word processor. It is performed on our C-Language Source code before the conversion to "Machine Code" takes place.

Use the Arduino IDE Software (use hardware setup from pg36, for LED)
1.Save as "pre processor"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = pre processor
#define LED_PIN 9

void setup(){
  pinMode(LED_PIN,OUTPUT);
}
void loop(){
  digitalWrite(LED_PIN,HIGH);
  delay(500);
  digitalWrite(LED_PIN,LOW);
  delay(500);
}
```

Watch the LED, same as the "classic blink" Program

```
#define LED_PIN 9
void setup() {
  pinMode(LED_PIN, OUTPUT);
}
```

A conversion will take place and the C-Language Compiler will compile the code below instead

```
void setup(){
  pinMode(9, 1);
}
```

#define LED_PIN 9
The C Language Pre-processor system will Search and Replace all the occurance of "LED_PIN" in our Program with "9"

Why "OUTPUT" is converted into "1" ?
There is a pre-processor directive **#define OUTPUT 1** code inside a file name "Arduino.h". So it will also Search and Replace "OUTPUT" with "1"
The "Arudino.h" file can be found in the folder below
<IDE install folder>/hardware/arduino/avr/cores/arduino/Arduino.h
#define HIGH 1 and #define LOW 0 can also be found there.

Why is Arduino.h file contents added into our Program ?
This is done by the pre-processor directive **#include <Arduino.h>**, this code is inside the main.cpp file (The same file where our main() function is coded).

Apart from **#define** and **#include**, We also have these common pre-processor directives (we will not be looking at all of them for now)

#undef	Undefines a #define
#ifdef	Returns true if #define exist
#ifndef	Returns true if #define does not exist
#if	Condition is true.
#else	Condition is false
#elif	Condition is combination of #else and #if
#endif	Condition ends
#error	Send compiler error to output screen
#pragma	Special Commands to the Compiler

Lets make a simple Device with our C-Language and Arduino Uno, with the limited C-Language Codes from our lessons so far.

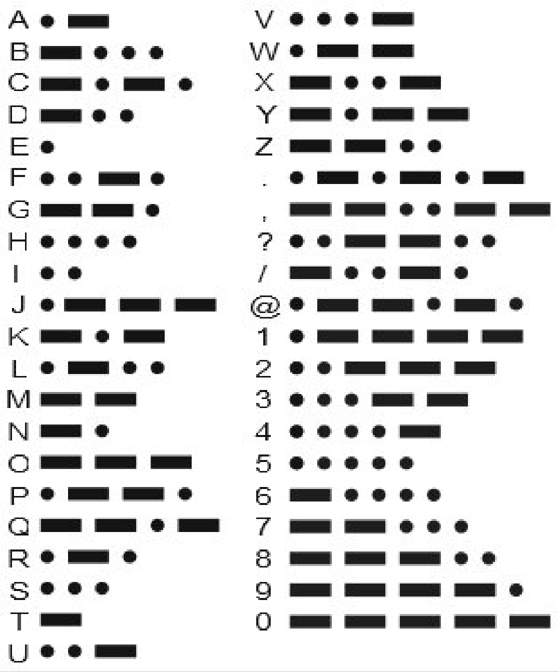
Use the Arduino IDE Software (use hardware setup from pg3, LED+Resistor)
1.Save as “morse code SOS LED”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = morse code SOS LED
#define LED_PIN         9
#define MC_DOT          300
#define MC_DASH         900  // DASH = DOT x 3
#define MC_PART_SPACE   300  // PART SPACE = DOT x 1
#define MC_LETTER_SPACE 900  // LETTER SPACE = DOT x 3
#define MC_WORD_SPACE   2100 // WORD SPACE = DOT x 7

void setup() {
 pinMode(LED_PIN,OUTPUT);
}
void loop() {
  mcLetter_S();digitalWrite(LED_PIN,LOW);delay(MC_LETTER_SPACE);
  mcLetter_O();digitalWrite(LED_PIN,LOW);delay(MC_LETTER_SPACE);
  mcLetter_S();digitalWrite(LED_PIN,LOW);delay(MC_WORD_SPACE);
}
void mcLetter_S() {
  digitalWrite(LED_PIN,HIGH); delay(MC_DOT);
  digitalWrite(LED_PIN,LOW);  delay(MC_PART_SPACE);
  digitalWrite(LED_PIN,HIGH); delay(MC_DOT);
  digitalWrite(LED_PIN,LOW);  delay(MC_PART_SPACE);
  digitalWrite(LED_PIN,HIGH); delay(MC_DOT);
}
void mcLetter_O() {
  digitalWrite(LED_PIN,HIGH); delay(MC_DASH);
  digitalWrite(LED_PIN,LOW);  delay(MC_PART_SPACE);
  digitalWrite(LED_PIN,HIGH); delay(MC_DASH);
  digitalWrite(LED_PIN,LOW);  delay(MC_PART_SPACE);
  digitalWrite(LED_PIN,HIGH); delay(MC_DASH);
}
```

Watch the LED, We are sending out Morse Code, S-0-S with our LED

International Morse Code

1 Dash = 3 Dots
Space between parts of same letter = 1 Dot
Space between letters = 3 Dots
Space between words = 7 Dots



A	• ▬	V	• • • ▬
B	▬ • • •	W	• ▬ ▬
C	▬ • ▬ •	X	▬ • • ▬
D	▬ • •	Y	▬ • ▬ ▬
E	•	Z	▬ ▬ • •
F	• • ▬ •	.	• ▬ • ▬ • ▬
G	▬ ▬ •	,	▬ ▬ • • ▬ ▬
H	• • • •	?	• • ▬ ▬ • •
I	• •	/	▬ • • ▬ •
J	• ▬ ▬ ▬	@	• ▬ ▬ • ▬ •
K	▬ • ▬	1	• ▬ ▬ ▬ ▬
L	• ▬ • •	2	• • ▬ ▬ ▬
M	▬ ▬	3	• • • ▬ ▬
N	▬ •	4	• • • • ▬
O	▬ ▬ ▬	5	• • • • •
P	• ▬ ▬ •	6	▬ • • • •
Q	▬ ▬ • ▬	7	▬ ▬ • • •
R	• ▬ •	8	▬ ▬ ▬ • •
S	• • •	9	▬ ▬ ▬ ▬ •
T	▬	0	▬ ▬ ▬ ▬ ▬
U	• • ▬		

With the limited C-Language codes from our lessons so far, We are now able to make a device that sends out S-0-S in Morse Code with our LED and Arduino Uno.

A simple blinking LED is not entirely useless, it can even save lives (with our simple Arduino-LED S-0-S device)

DataTypes and Working Memory (SRAM Memory)

Like any other Memory, SRAM Memory is also a long logical sequence of “0”s and “1”s. We do not use all of the SRAM Memory at one go. We use the SRAM Memory in smaller Memory Blocks known as DataTypes.

A DataType is a block of Memory from the SRAM Memory with a specific size and usage. Based on the DataType used, the C-Language will do automatic conversion between the data used (numbers, characters and symbols) in our Program and the actual binary storage in the SRAM Memory .

The basic DataTypes in Standard C-Language are char, int, short, long, float and double. I am sure some of us have seen more than that, this is because C-Language allows us to make more DataTypes from the basic DataTypes.

We will just focus on the basic DataTypes (There are 3 groups of DataTypes)

“unsigned” DataTypes. There is an “unsigned” keyword infront of the DataType They can only store **positive numbers.** They are stored in **SRAM** using direct binary representation of the number.

DataType	Memory Block	Decimal Value Range
unsigned char	8-bits (1 byte)	0 to 255
unsigned int	16-bits (2 bytes)	0 to 65,535
unsigned short	16-bits (2 bytes)	0 to 65,535
unsigned long	32-bits (4 bytes)	0 to 4,294,967,295

“signed” DataTypes. We can put the “signed” keyword infront of the DataType, however it is optional (we just leave it empty). They can store **positive and negative numbers.** They are automatically stored in **SRAM** using twos complement binary representation of the number with **most left bit used as sign bit.**

DataType	Memory Block	Decimal Value Range
char	8-bits (1 byte)	-128 to 127 (0 to 127 is used with a an ASCII representation)
int	16-bits (2 bytes)	-32,768 to 32,767
short	16-bits (2 bytes)	-32,768 to 32,767
long	32-bits (4 bytes)	-2,147,483,648 to 2,147,483,647

Floating point numbers (float and double). These number have decimal point in it. They can store **positive and negative numbers.** They are automatically stored in **SRAM** using IEEE 754 Floating Point Standard binary representation of the number.

DataType	Memory	Decimal Value Range
float	32-bits (4 bytes)	1.2E-38 to 3.4E+38 (6 decimal places)
double	64-bits (8 bytes)	2.3E-308 to 1.7E+308 (15 decimal places)

In Atmega328 micro-controller, both float and double are same 32-bits (4 bytes).

Use the Arduino IDE Software
1.Save as “basic_datatypes”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = basic_datatypes
void setup() {
char   myChar   =65; // we can also code myChar = 'A';
int    myInt    =65;
int    myIntN   =-65;
short  myShort  =65;
long   myLong   =65;
float  myFloat  =65.0;
double myDouble =65.0;

  Serial.begin(9600);
  Serial.print("\n\nBasic DataTypes");

  Serial.print("\n\nchar = ");Serial.print(myChar);
  Serial.print("\n");Serial.print(sizeof(myChar));
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=(sizeof(myChar)*8)-1; i>=0; i--) {
    Serial.print((myChar >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n\nint = ");Serial.print(myInt);
  Serial.print("\n");Serial.print(sizeof(myInt));
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=(sizeof(myInt)*8)-1; i>=0; i--) {
    Serial.print((myInt >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\nint = ");Serial.print(myIntN);
  Serial.print("\n");Serial.print(sizeof(myIntN));
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=(sizeof(myIntN)*8)-1; i>=0; i--) {
    Serial.print((myIntN >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n\nshort = ");Serial.print(myShort);
  Serial.print("\n");Serial.print(sizeof(myShort));
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=(sizeof(myShort)*8)-1; i>=0; i--) {
    Serial.print((myShort >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n\nlong = ");Serial.print(myLong);
  Serial.print("\n");Serial.print(sizeof(myLong));
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=(sizeof(myLong)*8)-1; i>=0; i--) {
    Serial.print((myLong >> i) & 0x1);Serial.print(" ");
  }
  Serial.print("\n\nfloat = ");Serial.print(myFloat,1);
  Serial.print("\n");Serial.print(sizeof(myFloat));showFloatBits(myFloat);
  Serial.print("\n\ndouble = ");Serial.print(myDouble,1);
  Serial.print("\n");Serial.print(sizeof(myDouble));showFloatBits(myDouble);
}
void loop(){}
void showFloatBits(float myFloat) {
union {
  float f;
  uint32_t u;
} memory;
  memory.f = myFloat;
  Serial.print("-byte, binary in SRAM Memory = ");
  for (int i=31; i>=0; i--) {
    Serial.print((memory.u >> i) & 0x1);Serial.print(" ");
  }
}
```

Use the Serial Monitor to see output

Variable are reference to a Memory Block of a DataType. Before we can use a Variable we must define them with a DataType. In C-Language we can define Variables at 3 places.

1. Outside a function (Known as Global Variables)
2. Inside a function (Known as Local Variables)
3. Function Parameters (usage is same as Local variables)

Use the Arduino IDE Software
1. Save as “variable scope”
2. Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = variable scope
const char myVarX = 'X';
char myVar1;
void setup() {
char myVar2;
  myVar1 = 'G';
  myVar2 = 'S';
  // myVarX = 'M'; // the C-Compiler will block any attempt to use this
  Serial.begin(9600);
  Serial.print("\n\nin setup(),myVar1 declared outside functions,myVar1=");
  Serial.print(myVar1);
  Serial.print("\n\nin setup(),myVar2 declared inside,myVar2=");
  Serial.print(myVar2);
  myFunction1();
  myFunction2('2');
}
void loop(){}
void myFunction1() {
char myVarL;
  myVarL = '1';
  Serial.print("\n\nin myFunction1(),myVarL declared inside,myVarL=");
  Serial.print(myVarL);
  Serial.print("\nDisplaying myVar1 from myFunction1,myVar1=");
  Serial.print(myVar1);
}
void myFunction2(char myVarL) {
  Serial.print("\n\nin myFunction2(),myVarL declared in parameter,myVarL=");
  Serial.print(myVarL);
  Serial.print("\nDisplaying myVar1 from myFunction2,myVar1=");
  Serial.print(myVar1);
}
```

Use the Serial Monitor to see output, this example is for char, it will work the same for other DataTypes

char myVar1
This is a global variable, defined outside functions. It can be accessed and updated from any parts of our Program

char myVar2
This is a local variable inside setup() function, defined inside setup() function. It can be accessed from inside the setup() function

char myVarL
myVarL Variable appeared in myFunction1 and myFunction2, both are local to the function they are defined, They can have the same name because they are local to where they are defined, not accessible outside the function.

const char myVarX = ‘X’;
We can also add a keyword “const” in front of the Variable definition, this will make the Variable “ReadOnly” where it can only accept a value during variable definition.

Any attempt to use a Local Variable outside their scope or to change a Readonly Variable will be stopped by the Compiler

Earlier we work with an Output Device, the LED. This time we have an Example Project using an Input Device, Tactile button with the Arduino Uno (see page 36 for Tactile Switch/Arduino Uno “A0” connection)

Use the Arduino IDE Software (use hardware setup from pg36, for tactile)
1.Save as “tactile A0”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = tactile A0
void setup() {
  Serial.begin(9600);
}
void loop(){
unsigned int pinValue;

  pinValue = digitalRead(A0);
  Serial.print("Digital Value = ");
  Serial.print(pinValue);

  pinValue = analogRead(A0);
  Serial.print(", Analog Value = ");
  Serial.print(pinValue);

  Serial.print("\n");
}
```

Use the Serial Monitor to see input values when button is pressed and released

When we work with OUTPUT device (the LED) in our earlier Project, we need to code pinMode(9, OUTPUT) to tell the micro-controller that that pin will be an OUTPUT pin.

For INPUT pin, we do not need to code anything because INPUT is the default value.

The tactile switch is connected to “A0” of the Arduino Uno board, which is a pin from the Atmega328 micro-controller that can function as both Digital (gives either 1 or 0 only dependig on the voltage on the Input Device) and Analog pin (gives a number range from 1 to 1023 depending on the voltage on the Input device).

Why 0 or 1 for Digital Reading ? pinValue = digitalRead(A0);

Digital is either 1 or 0. When button is pressed, the pin gets 5V input or something close to %V, it is considered HIGH Voltage (translated to 1), when the button is released, the pin gets 0V or something close to 0V, it is consided LOW Voltage (translated to 0)

Why 0 to 1023 for Analog Reading ? pinValue = analogRead(A0);

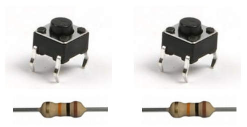
Inside Atmega328 micro-controller, there are Analog Digital Converter (ADC) for some pins (A0 is one of them). ADC in Atmega328 micro-controller uses 10-bit Register Memory.

10-bits stores 0000000000 to 1111111111 (0 to 1023 in decimal)

IF voltage at the pin is 5V from input device, the value will be 1023
IF voltage at the pin is 0V from input device, the value will be 0

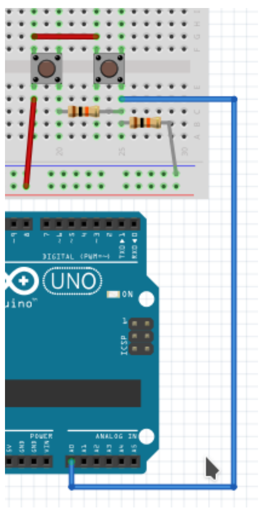
if voltage is in-between 0V and 5V, the micro-controller will calculate for us.
For example: if the voltage is 3V, the value will be (3/5) x 1023 = 614

Lets change our Hardware a little bit by adding in another tactile switch with a Resistor and a jumper wire, we can continue with the same Program. No need to upload again. Just watch the Serial Monitor while pressing each button.



This is the the same Program uploaded earlier

```
// program name = tactile2 A0
void setup() {
  Serial.begin(9600);
}
void loop(){
unsigned int pinValue;

  pinValue = digitalRead(A0);
  Serial.print("Digital Value = ");
  Serial.print(pinValue);

  pinValue = analogRead(A0);
  Serial.print(", Analog Value = ");
  Serial.print(pinValue);

  Serial.print("\n");
}
```



Use the Serial Monitor to see input values when button is pressed and released

When we press the first tactile switch, we will get a different Analog Reading compared to pressing the second tactile switch. Additional Resistor gives different Current at both switch, effecting the Voltage on both tactile switch.

Means we can tell the which tactile switch is being pressed.

Which also means, we can have many tactile switch on a single pin (just add more resistor and tactile switch which is not possible with digital Pins, for digital pins we can only have 1 button for 1 pin).

This example shows just 2 tactile button, you can have more if you wish, by adding more tactile switch+resistor+jumper wires. This will save us alot of digital pins.

Use tactile switch as switch with LED. Simple Coding

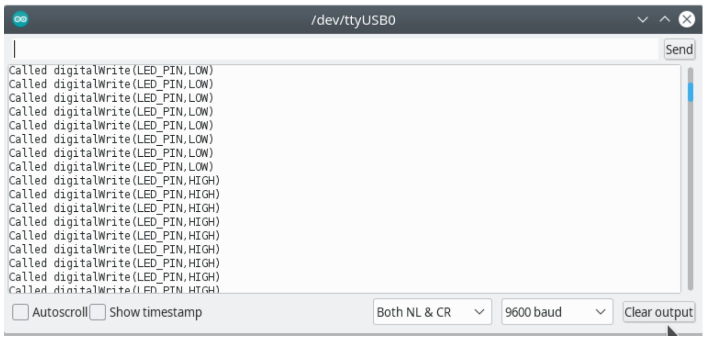
Use the Arduino IDE Software
1.Save as “simple LED switch”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = simple LED switch
#define LED_PIN     9
#define BUTTON_PIN  A0

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

void loop() {
  if (digitalRead(BUTTON_PIN) == HIGH) {
    Serial.println("Called digitalWrite(LED_PIN,HIGH)");
    digitalWrite(LED_PIN,HIGH);
  } else {
    Serial.println("Called digitalWrite(LED_PIN,LOW)");
    digitalWrite(LED_PIN,LOW);
  }
}
```

Watch the LED when tactile switch is pressed and released

When we press the tactile switch, LED will light up, as expected. When we released the tactile switch, LED will turn OFF. A simple momentary switch.

While watching the LED when tactile switch is pressed and released, now we open up the Serial Monitor to see what is going on with our Program.



```
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,HIGH)
```

Although our LED and tactile works as expected but the actual excecution of Program codes is awful.

If we look at the Serial Monitor, even when the tactile switch is not pressed and LED stays OFF, the Program kept on calling digitalWrite() function to turn off LED and when tactile button pressed down, the Program kept on calling the digitalWrite() function to turn on LED even though the LED is already switched-ON.

So we need to do some improvement to our Program, otherwise we will be wasting electricty and “wear out” our hardware fast.

Use tactile switch as switch with LED. Coding Improved

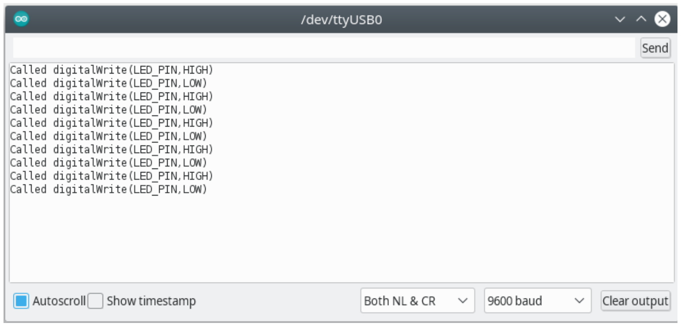
Use the Arduino IDE Software (use hardware setup from pg36, for LED+tactile)
1.Save as "improved LED switch"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = improved LED switch
#define LED_PIN      9
#define BUTTON_PIN   A0

uint8_t ledState = 0;

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

void loop() {
  if (digitalRead(BUTTON_PIN) == HIGH) {
    if (ledState == 0 ) {
      Serial.println("Called digitalWrite(LED_PIN,HIGH)");
      digitalWrite(LED_PIN,HIGH);
      ledState = 1;
    }
  } else {
    if (ledState == 1) {
      Serial.println("Called digitalWrite(LED_PIN,LOW)");
      digitalWrite(LED_PIN,LOW);
      ledState = 0;
    }
  }
}
```

Watch the LED when tactile switch is pressed and released

When we press the tactile switch, LED will light up, as expected. When we released the tactile switch, LED will turn OFF. On the surface this Program and the earlier Program works exactly the same. BUT...

Now, lets open up our Serial Monitor to see what is going on inside the Program while the LED when tactile switch is pressed and released.



```
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,LOW)
Called digitalWrite(LED_PIN,HIGH)
Called digitalWrite(LED_PIN,LOW)
```

As we can see from our Serial Monitor now, our Program only call the instruction code to switch ON and OFF LED, digitalWrite() once for each time when the button is pressed or released. The program is far more efficient than the earlier one. Less execution, means less power used and also means less activity on our hardware(hardware will "wear out" very much slower)

C-Language Notes:

uint8_t This is a new DataType derieved from our basic C-Language DataTypes. It is "unsigned int" DataType that uses 8-bits of Memory.

In this example, we make a Toggle Switch from our Tactile Switch.

Use the Arduino IDE Software (use hardware setup from pg36, for LED+tactile)
1.Save as “tactile toggle switch”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = tactile toggle switch
#define LED_PIN     9
#define BUTTON_PIN A0

uint8_t oldButtonState = 0;
uint8_t ledState = 0;

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

void loop() {
uint8_t newButtonState;
  newButtonState = digitalRead(BUTTON_PIN);
  if (newButtonState != oldButtonState) { // Button State Change
    oldButtonState = newButtonState;
    delay(30); // this is to filter eletro-mechanical "noise"
    if (newButtonState == HIGH) {
      Serial.print("\nButton Pressed Down. LED State = ");
      Serial.print(ledState);
      ledState = (ledState==1)?0:1;
      digitalWrite(LED_PIN,ledState);
      Serial.print(", changed to ");Serial.print(ledState);
    }
  }
}
```

Watch the LED and see the Serial Monitor to see button is pressed

When button is pressed down, the program will toggle between switching the LED ON and OFF.

An Array variable is a varaiable that can store multiple values. Here, we just look at **Basic Array**, simple **one dimensional single DataType**, **fixed element Array**. Array are be **defined** and **accessed** with the **"[]"** symbol pair.

Below is a common way to define an Array variable. Below we **define an Array** Variable with 5 elements.

```
// define an Array of 5 elements of char DataType, elements to be filled later
char myCharArray[5];

// define an Array of 5 elements of char DataType with default values
char myCharArray[] = {'A','B','C','D','E'};
```

There access or update an elements in an array, we used an **index** within the "[]" symbol pair.
For example: **myCharArray[3]** means we are accessing the **4th element** of the myCharArray variable. Why 4th and and not 3rd ? This is because the **Array index starts from 0.** (first element is myCharArray[0])

```
char myCharArray[] = {'A','B','C','D','E'};
To Access, myCharArray[3]; // gives you  'D'
To Update, myCharArray[3] = 'X'; // Change 'D' to 'X'
```

Use the Arduino IDE Software
1.Save as "array basic"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = array basic
void setup() {
char myCharArray[] = {'A','B','C','D','E'};

  Serial.begin(9600);
  Serial.print("\n\nmyCharArray Element 4, index 3 = " );
  Serial.print(myCharArray[3]);

  myCharArray[3] = 'X';
  Serial.print("\n\nmyCharArray[3] = 'X';" );

  Serial.print("\n\nmyCharArray Element 4, index 3 = " );
  Serial.print(myCharArray[3]);

}
void loop() {}
```

Use Serial Monitor to see Output

For multiple dimension, simply specify more "[]" symbol pair

```
char myCharArray[2][2]; // means 2 dimensions, 2 by 2

char myCharArray[][] = {{'1','2},{'1','2'}} // means 2 dimensions, 2 by 2 with
default values

char myCharArray[2][2][2]; // means 3 dimensions, 2 by 2 by 2
and so on...
```

The C-Language Array can be made to handle dynamic length array and multiple-DataTypes Array. To do that, we needs more advanced C-Language programming tecnniques (Memory Allocation and Pointers). For now, we stick to the basics.

This works similar to our earlier Morse Code SOS Program but using an Array. Simple practical use of an Array.

Use the Arduino IDE Software (use hardware setup from pg36, for LED)
1.Save as “array basic SOS”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = array basic SOS
#define LED_PIN     9
#define MC_DOT     300
#define MC_DASH    900  // DASH = DOT x 3
#define MC_PART    300  // PART SPACE = DOT x 1
#define MC_LETTER  900  // LETTER SPACE = DOT x 3
#define MC_WORD    2100 // WORD SPACE = DOT x 7

unsigned long mcMessage[] = { // SOS message
MC_DOT,MC_PART,MC_DOT,MC_PART,MC_DOT,MC_LETTER,
MC_DASH,MC_PART,MC_DASH,MC_PART,MC_DASH,MC_LETTER,
MC_DOT,MC_PART,MC_DOT,MC_PART,MC_DOT,MC_LETTER,
0
};

uint8_t ledState = 0;

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

void loop() {
  mcOutput();
}

void mcOutput() {
int aIndex = 0;
  while (mcMessage[aIndex] != 0) {
    ledState = (ledState==1) ?0:1;
    digitalWrite(LED_PIN,ledState); delay(mcMessage[aIndex]);
    aIndex++;
  }
  digitalWrite(LED_PIN, MC_WORD);
}
```

Watch the LED, sending SOS morse code like our earlier example.

In this example, we place time delay inside an array variable for each morse-code parts, then later we simply retrieve data from our Array for output

In C-Language, there is no native "string" DataType. One way is to use the char Array to represent strings.

'' symbol pair means a single character, we can only put a single character between the '' pair.

We can store single character into a char DataType variable or an element in an char Array DataType.

```
// Example, A Simple char in an char Array
char myCharArray[10];
myCharArray[0] = 'A';
```

"" symbol pair means a string, we can have multiple characters between the "" pair.

Although we use "" symbol pair to represent a "string", it is just a sequence of multiple char in an array, terminated by a null character '\0' or ASCII 0.

The array used by "string" is char DataType Array.

```
// for example, to store a string "AB"
char myCharArray[10];

myCharArray[0] = 'A';
myCharArray[1] = 'B';
MyCharArray[2] = '\0'; // '\0' is a null character
```

Use the Arduino IDE Software
1.Save as "string basic"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = string basic
void setup() {
char myCharArray[10];
char myDefaultArray[] = "XY";

  Serial.begin(9600);
  myCharArray[0] = 'A';
  myCharArray[1] = 'B';
  myCharArray[2] = 'C';
  myCharArray[3] = '\0';
  Serial.print("\n\---Start --- store string ABC in a char Array = ");
  Serial.print(myCharArray);
  Serial.print("\nNumber of elements in Array = ");
  Serial.print(sizeof(myCharArray));

  Serial.print("\n\ndefine n store default string XY in char Array = ");
  Serial.print(myDefaultArray);
  Serial.print("\nNumber of elements in Array = ");
  Serial.print(sizeof(myDefaultArray));
}
void loop() {}
```

Use Serial Monitor to see Output

char myCharArray[10];
This above code creates an char Array DataType with 10 elements, to be filled later. In our example above, not all 10 elements are used for ABC "string".

char myDefaultArray[] = "XY";
This above code creates an char Array DataType with 3 elements,
myDefaultArray[0] = 'X', myDefaultArray[1] = 'Y', myDefaultArray[2] = '\0'

Use the Arduino IDE Software
1.Save as “string array dump”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = string array dump
void setup() {
char myCharArray[20];

  Serial.begin(9600);

  buildString(myCharArray, "THIS IS A STRING");
  Serial.print("\n\nExample 1: myCharArray is now = ");
  Serial.print(myCharArray);
  serialArrayDump(myCharArray);

  // Build another string
  buildString(myCharArray, "NEW STRING");
  Serial.print("\n\nExample 2: myCharArray is now = ");
  Serial.print(myCharArray);
  serialArrayDump(myCharArray);

  // Cut off at index 6
  myCharArray[6] = 0;
  Serial.print("\n\nExample 3: null at index 6, myCharArray is now = ");
  Serial.print(myCharArray);
  serialArrayDump(myCharArray);
}
void loop() {}


void buildString(char* arrayString, char inputString[]) {
int numChars;
int iCtr = 0;
  numChars = strlen(inputString);
  while (iCtr < numChars) {
    arrayString[iCtr] = inputString[iCtr];
    iCtr++;
  }
  arrayString[iCtr] = '\0';
}

void serialArrayDump(char* myCharArray) {
char nullFound = 'N';
  Serial.print("\nArray Dump");
  for (int i=0; i<20;i++ ) {
    Serial.print("\n[");Serial.print(i);Serial.print("] =");
    Serial.print(myCharArray[i]); Serial.print(", ASCII = ");
    Serial.print((int) myCharArray[i]);
    if (myCharArray[i] == 0 and nullFound == 'N') {
      Serial.print("<--- string ends here, after this ignored");
      nullFound = 'Y';
    }
  }
}
```

Use Serial Monitor to see output

In this example, we create an array of 20 elements. We then load the array with two different strings, “THIS IS A STRING”, and then “NEW STRING”.

Later a null character is placed into the middle the char Array, the “NEW STRING” string will be adjusted according to the position of the null character ‘\0’ or ASCII 0

This Program also does an “array dump” in Serial Monitor for us to see what is in each elements of the array.

A **struct** variable is similar to an Array but can have multiple elements of different DataTypes. **Instead of index, struct elements** are be **accessed** using the **"."** symbol with the **element name**. The elements in struct is called **"members"**, members are **variables of any DataTypes**.

Below is a common way to define an Array variable. Below we **define a simple Struct** 2 elements.

```
struct colRow {
  char columnName;
  int data;
};
```

Use the Arduino IDE Software
1.Save as "struct basic"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = struct basic
void setup() {
struct colRow {
  char columnName;
  int data;
};
  Serial.begin(9600);

  struct colRow myColRow; // the struct variable is myColRow

  // Assign value into the members
  myColRow.columnName = 'A';
  myColRow.data = 123;

  Serial.print("\n\n--- start ---- Accessing the struct ");

  Serial.print("\nColumn name = ");
  Serial.print(myColRow.columnName);
  Serial.print("\ndata = ");
  Serial.print(myColRow.data);

  Serial.print("\n\nChange values stored");
  myColRow.columnName = 'B';
  myColRow.data = 456;;

  Serial.print("\n\nChanged Column name = ");
  Serial.print(myColRow.columnName);
  Serial.print("\nChanged data = ");
  Serial.print(myColRow.data);
}
void loop() {}
```

Use the Serial Monitor to see Output

In this example, we are looking a very basic struct, with two members. columnName and data, both from different DataType.

The strength of C-Language struct is, The struct members can be any DataType, including char Array (to store strings) and any DataTypes including other struct DataType.

The struct can also be elements of an Array,
struct colRow myColRow[5]; // means an array of 5 elements of struct colRow

With that,we get a worksheet 5 column x 5 rows).

The struct can even can be used to handle more complicated situation, like the multi-Dimensional, multi-DataTypes , it is up to us how to mix and match the DataTypes.

An **union** variable is similar to a struct, also with elements of different DataTypes. **union elements** are also **accessed** using the **"."** symbol with the **element name**. The elements in union is called **"members"**, members are **variables of any DataTypes**. Until now, union and struct looks the same.

Now the difference.

union use one single memory space for all its members, while **struct uses multiple memory space** for each of its members.

The single Union memory used is based the largest DataType in its member list.

Use the Arduino IDE Software
1.Save as "union basic"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program Name = union basic
void setup (void) {
float myVar;
union memory {
  float f;    // uses 32-bit
  uint32_t i; // uses 32-bit
};

  union memory myMemory;

  myVar = 0.078125;
  myMemory.f = myVar; // load value into member f
  // Both myMemory.f and myMemory.i shares the same memory space

  Serial.begin(9600);
  Serial.print("\n\n--- Start --- float value = ");Serial.print(myMemory.f,6);

  Serial.print("\n\nTo see each bit stored in float IEEE 754 format");
  Serial.print("\nExtract each bit from 32-bit int DataType  = ");
  for (int y=31; y>=0; y--) {
    Serial.print((myMemory.i >> y) & 1);Serial.print(" ");
  }
}
void loop (void) {}
```

Use the Serial Monitor to see Output

In this example, we are looking a practical usage of an union. We wanted to see what how the float is stored in SRAM Memory. We cannot take out each bit from a float DataType for display on the Serial Monitor but we can do it with an int DataType.

Therefore, we use an union with 2 members, float and 32-bit int DataType, Since both of the shares the 32-bit memory space, we can use the 32-bit int to extract the actual binary storage of float in the same memory space.

Of cos there are other usage for union. For example, we use it to do data conversion, use it to save memory, the same memory space can be used for multiple purpose and etc. It is up to us how we want to use it.

An **enum** variable is similar to a union, it uses a single SRAM Memory space, with one or multiple “tags” as elements. Each “tags” name represents a whole number. Tag number, starting from 0, auto-increment by 1 for the next “tag” element. The starting value for a “tag” element can be set by assigning a number during the enum definition, the following elements will auto-increase by 1. Look the example Program below to get a clearer picture.

Use the Arduino IDE Software
1.Save as “enum basic”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = enum basic
void setup() {
// enum daysInWeek {sun,mon,tue,wed,thu,fri,sat}; // ideal version
enum daysInWeek {sun,mon,tue=52,wed,thu=-5,fri,sat}; // messed up version

  Serial.begin(9600);

  enum daysInWeek myDay;

  Serial.print("\n\n--- Start --- Accessing enum");
  Serial.print("\n\nNo tag is assigned to myDay variable, myDay = ");
  Serial.print(myDay);

  // first element tag represents zero
  // unless assigned a number during enum definition
  myDay = mon;
  Serial.print("\n\nmyDay assigned with mon tag = ");
  Serial.print(myDay);

  myDay = wed;
  Serial.print("\n\nmyDay assigned with wed tag = ");
  Serial.print(myDay);

  myDay = sat;
  Serial.print("\n\nmyDay assigned with sat tag = ");
  Serial.print(myDay);
}
void loop() {}
```

Use the Serial Monitor to see Output

First we must define a enum DataType, just “tag” names one after another seperated by comma ,

enum daysInWeek {sun,mon,tue,wed,thu,fri,sat};

To use, we define a variable for that enum DataType

enum daysInWeek myDay;
myDay is now an enum DataType variable.

We can now use the **myDay** variable by assigning “tag” name to it . The tag must be from the enum definition.

for example, **myDay = mon;**
Since mon tag is the second element in the enum, myDay will become 1, because the sun tag in the enum represents 0.

Now we “messed up” the tags in the enum definition a little bit, on the **tue** and **fri** tag. **enum daysInWeek {sun,mon,tue=52,wed,thu,fri=-5,sat};**

The **tue** tag **represent 52**, tags after that will be increased by 1 one after another. That is why when we assign myDay variable with wed tag, “myDay = wed;” myDay variable will become 53. If we did not “messed up” tue tag with 52, myDay variable will be 3

Pass variables to other functions by “value”

Only value got transferred to called function, sent into receiving variable inside the called function parameter(s).

Use the Arduino IDE Software
1.Save as “pass by value”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = pass by value
void setup() {
char myChar = 'X';

  Serial.begin(9600);
  Serial.print("\n\n--- Start --- inside setup function");
  Serial.print("\nmyChar value = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);

  myFunction('Y');

  myOtherFunction(myChar);

  Serial.print("\n\nback to setup function");
  Serial.print("\nmyChar value(still the same) = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);
}

void loop() {}

void myFunction(char myChar) {
  Serial.print("\n\ninside myFunction");
  Serial.print("\nmyChar value = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);
}

void myOtherFunction(char otherChar) {
  Serial.print("\n\ninside myOtherFunction");
  Serial.print("\notherChar value = "); Serial.print(otherChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&otherChar);
}
```

Use Serial Monitor to see output

In this program, there are 3 separate variables
myChar inside the setup()
myChar inside the myfunction()
otherChar inside the myOtherFunction()

You can see all three of them are using different Memory Address from the Serial Monitor Output

```
void myFunction (char myChar) {
//...
}
```

When char myChar is defined as the parameter (within the bracket pair (), when called from other function as myFunction(‘Y’); It is similar to char myChar = ‘Y’,

Pass variables to other functions by “reference” or “Memory Address”

Memory Address of Variable is passed to called function, called function variable will receive “Memory Address” instead of “value” in its parameter(s).

Use the Arduino IDE Software
1.Save as “pass by reference”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = pass by reference
void setup() {
char myChar = 'X';

  Serial.begin(9600);
  Serial.print("\n\n---Start --- inside setup function");
  Serial.print("\nmyChar value = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);

  myFunction(&myChar);

  Serial.print("\n\nback to setup function");
  Serial.print("\nmyChar value(changed) = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);

  myOtherFunction(&myChar);

  Serial.print("\n\nback to setup function");
  Serial.print("\nmyChar value(changed) = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)&myChar);
}

void loop() {}

void myFunction(char *myChar) {
  Serial.print("\n\ninside myFunction");
  Serial.print("\nmyChar start value = "); Serial.print(*myChar);
  Serial.print("\nMemory Address = "); Serial.print((int)myChar);

  *myChar = 'Z'; // Change value
}

void myOtherFunction(char *sameChar) {
  Serial.print("\n\ninside myOtherFunction");
  Serial.print("\nsameChar start value = "); Serial.print(*sameChar);
  Serial.print("\nMemory Address = "); Serial.print((int)sameChar);

  *sameChar = 'Y'; // Change value
}
```

Use Serial Monitor to see output

In this program, there is just 1 variable to store value, the myChar variable from the setup() function. The variables inside the myFunction() and myOtherFunction() parameters are variable that stores Memory Address (or known as *pointers)

```
void myFunction(char *myChar) {
//...
}
```

char *myChar is a pointer variable. myChar stores Memory Address of a char DataType, it does not store the actual data. To access to the data, we specify the pointer symbol “*****” infront of the pointer name, ***myChar**

myFunction(&myChar);
The symbol “**&**” is to extract the Memory Address of a variable

Pass array variables to other functions

Array Variable can only be passed by reference. Array Variable itself is a Pointer Variable that holds a Memory Address.

Use the Arduino IDE Software
1.Save as “pass array”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = pass array
void setup() {
char myCharArray[4] = {'A','B','C','\0'};

  Serial.begin(9600);
  Serial.print("\n\n---Start --- inside setup function");
  Serial.print("\nmyCharArray value = "); Serial.print(myCharArray);
  Serial.print("\nMemory Address = "); Serial.print((int) myCharArray);

  myFunction(myCharArray);

  Serial.print("\n\nback to setup function");
  Serial.print("\nmyCharArray value = "); Serial.print(myCharArray);
  Serial.print("\nMemory Address = "); Serial.print((int) myCharArray);
}

void loop() {}

void myFunction(char myChar[]) {
  Serial.print("\n\ninside myFunction");
  Serial.print("\nmyChar start value = "); Serial.print(myChar);
  Serial.print("\nMemory Address = "); Serial.print((int) myChar);

  myChar[1] = 'X'; // Change value
}
```

Use Serial Monitor to see output

In this program, there is just 1 variable to store value myCharArray inside the setup() function. There is no need to use pointer symbol “*” and Memory Address “&” address symbol because the Array Variable is already a Memory Address.

myFunction(myCharArray);
To pass an array to an function, simply specify the Array variable name.

```
void myFunction(char myChar[]) { {
//...
}
```

The called function will receive the Memory Address from its array definition in its parameter variable, same Array DataType.

We can also do like this,

```
myFunction(“ABC”);
```

The called function must have the following receiving parameter variable,

```
void myFunction(char myChar[]) {
//...
}
```

Also means the char Array definition is `char myChar[] = {'A','B','C','\0'};`

*For struct and union, they are also handled the same way like array, they are also passed by reference

An Array variable has a fixed number of elements, Sometimes we may want to work with a variable that also have multiple elements but with variable elements, to do that we will need to use a feature from the C-Language. **Dynamic Memory Allocation and Pointers. Use these with special Care, this is famous for cuasing Operating System Memory Leak in Regular Computers.**

void *malloc(size_t size) and **void *calloc(size_t nitems, size_t size)**
- Both reserved a chunk of SRAM Memory, calloc() automatically turn all the bits into ‘0’ where malloc() just keep whatever that is there.

void free(void *ptr)
- Release memory back to SRAM (this one is often forgotten in big Programs, has caused alot of System run out of Memory and Crash)

void *realloc(void *ptr, size_t size)
- Changed the size of an existing reserved chunk of SRAM Memory

size_t is a positive whole number

Use the Arduino IDE Software
1.Save as “dynamic memory”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = dynamic memory
void setup() {
char *pChar;
int nElements;

  Serial.begin(9600);

  nElements = 5;
  pChar = (char*) malloc(nElements*sizeof(char));

  Serial.print("\n\n--- Start --- Allocating SRAM Memory for 5 char");
  Serial.print("\nLoad 3 elements into Array");
  pChar[0] = 'A'; pChar[1] = 'C'; pChar[2] = 'E';

  // Read from Memory, array style
  Serial.print("\nChar Array position[0] = ");Serial.print(pChar[0]);
  Serial.print("\nChar Array position[1] = ");Serial.print(pChar[1]);
  Serial.print("\nChar Array position[2] = ");Serial.print(pChar[2]);

  Serial.print("\n\nUpdate 3rd element with Z");
  // Load value into the Memory, Pointer style
  *(pChar+2*sizeof(char)) = 'Z';

  // Read from Memory, use pointer address offset position
  Serial.print("\nPointer position offset 2 from start = ");
  Serial.print(*(pChar+2*sizeof(char)));

  free(pChar);
}
void loop() {}
```

Use Serial Monitor to see Output

This is not a regular array but we can access the stored value like an Array or we can also use Memory Pointer to access to the stored value. The allocated Memory size can be changed by using realloc() function

Unfortunately, C-Language does not provide us with any function to retrieve the number of bytes the malloc() or calloc() has allocated for our pointer. We have to keep track of the memory reserved. There is a danger of us accessing outside of the intended memory allocation, into unknown SRAM Memory space and C-Language allows it (some people called it powerful, some people call it weakness). No matter what, make sure remember to execute the free() function once we are done with it.

Math symbols are used to perform mathematic operations on numbers in our Program.

```
C-Language Math Operation Symbols
* Multiply, a=b*c; a = result from b multiply c
% Modulus,  a=b%c; a = remainder of a division, b divided by c
/ Divide,   a=b/c; a = result from b divided by c
+ Plus,     a=b+c; a = result from b plus c
- Minus,    a=b-c; a = result from b minus c
```

We can do shortcut coding when using math operation symbols

```
a=a+b; can also be coded as a+=b; // you can do the same for other operations
```

Increase and Decrease number by 1

```
b++; // use original value, then increase by 1 after operation is completed
b--; // use original value, then increase by 1 after operation is completed
++b; // increase by 1 before operation starts
--b; // decrease by 1 before operation starts

b=4, c=7;
a = b++ + ++c; // results = 4+8 = 12;
```

We can use the bracket () pair symbol to enforce math operation precedence

```
a = 2; b=4, c=7;

a = (b+(c-a))*a;

Step 1. This will start with most inner bracket pair “()” c-a
Step 2. result of Step 1. plus b
Step 3. result of Step 2. multiply a
Step 4. result of Step 3. assigned to a
```

Relation/Logic Symbols are used to make decision based on certian conditions encounted during Program execution.

C-Language Relation and Logic Symbols

```
==, (a==b); a is equals to b ( exactly the same )
!=, (a!=b); a is not equal to b
<,  (a<b);  a is less than b
>,  (a>b);  a is more than b
<=, (a<=b); a is less or equal to b
>=, (a>=b); a is more or equal to b

If the comparison in the bracket is true,  returns 1
If the comparison in the bracket is false, returns 0

||, OR operator
(1 || 1); = 1
(1 || 0); = 1
(0 || 1); = 1
(0 || 0); = 0

&&, AND operator
(1 && 1); = 1
(1 && 0); = 0
(0 && 1); = 0
(0 && 0); = 0

!, NOT operator
!1; = 0
!0; = 1
```

Program controls symbols are used as part of C-Language instruction codes for various purpose.

C-Language Program Control Symbols

#	Used as C-Language Pre-processor directives
< >	A Pair is used for the C-Language Pre-processor #include directive
“ “	A Pair is used to specify a “string”
‘ ‘	A Pair is used to specify a char
( )	A Pair is used for function parameters and math operation precedence
{ }	A Pair to group a block of instruction codes
[ ]	A Pair to identify Array
//	Used for single line programmer notes/comments
/* */	A Pair for multiple line programmer notes/comments
;	Used to identify end of a instruction code operation
,	Used as separator for parameters, variables and others
:	Used by switch/case statement and also tenary “?” Operator
?	Used as tenary operator together with “:” symbol
.	Used as struct/union member identifier and in floating point numbers
\	Used to insert special characters into a string or char
=	Used as Variable assignment, value on the right assigned to the left

Memory Address and Pointers

&	return variable Memory Address
*	contains Memory Address (pointer DataType)

C-Language Bit Operation Symbols

&	bit AND Operation
|	bit OR Operation
^	bit XOR Operation
~	bit NOT Operation
<<	bit LEFT SHIFT Operation
>>	bit RIGHT SHIFT Operation

Use the Arduino IDE Software
1.Save as “if else”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program Name = if else
#define LDR_PIN A1

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

void loop(){
int pinValue;

  pinValue = analogRead(LDR_PIN);
  Serial.print("\nA1 Reading = ");Serial.print(pinValue);
  if (pinValue < 150) {
    Serial.print(" , Surroundings is DARK");
  } else {
    Serial.print(" , Surroundings is BRIGHT");
  }
}
```

Use Serial Monitor to see Output

The Serial Monitor will show the readings we get from connector “A1” on the Arduino Uno board. The reading depends on the LDR sensor device.

The value inside the if bracket “()” pair is a number, it is either **0** or **non-zero**.

if(0), will run the codes from the else portion
if(non-zero), will run the codes from the if portion

In this example our bracket () contains a comparison **if(pinValue < 200)**, **pinValue < 200** will give us **if(1) when true**, and **if(0) when false**.

Use the Arduino IDE Software
1.Save as "switch case"
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// Program name = switch case
#define LDR_PIN A1
#define LEVEL_0 150
#define LEVEL_1 300
#define LEVEL_2 450

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

void loop(){
int bLevel;
int pinValue;

  pinValue = analogRead(LDR_PIN);
  Serial.print("\nReading = ");Serial.print(pinValue);
  bLevel = calcLevel(pinValue);
  switch (bLevel) {
    case 1:
      Serial.print(" , Brightness Level 1");
    break;
    case 2:
      Serial.print(" , Brightness Level 2");
    break;
    case 3:
      Serial.print(" , Brightness Level 3");
    break;
    case 4:
      Serial.print(" , Brightness Level 4");
    break;
    default:
    break;
  }
}

int calcLevel(int pinValue) {
  if (pinValue < LEVEL_0 ) return 1;
  if (pinValue >= LEVEL_0 && pinValue < LEVEL_1) return 2;
  if (pinValue >= LEVEL_1 && pinValue < LEVEL_2) return 3;
  if (pinValue >= LEVEL_2 ) return 4;
}
```

Use Serial Monitor to see Output

The Serial Monitor will show the readings we get from connector "A1" on the Arduino Uno board. The reading depends on the LDR sensor device.

Based on the "A1" Reading, we make 4 different category, stored in a variable bLevel.

We use the bLeval variable with our switch-case(). From our code, we can see the switch-case() can have more than 2 options of codes to execute.

Use the Arduino IDE Software
1.Save as “for”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = for
#define LED_PIN 9
void setup() {
  pinMode(LED_PIN, OUTPUT);
  for (int i=0; i < 5; i++) {
    digitalWrite(LED_PIN, HIGH); delay(500);
    digitalWrite(LED_PIN, LOW); delay(500);
  }
}
void loop(){}
```

Watch LED blink 5 times

“for” will repeat codes within its curly bracket until the condition (i < 5) returns a 0 (when false, will return a 0).The for() can have a variable definition with a default value (run just once), a condition and a increment counter (repeated).

Use the Arduino IDE Software
1.Save as “while”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = while
#define LED_PIN 9
void setup() {
int i = 0;
  pinMode(LED_PIN, OUTPUT);
  while( i < 5 ) {
    digitalWrite(LED_PIN, HIGH); delay(500);
    digitalWrite(LED_PIN, LOW); delay(500);
    i = i+1;
  }
}
void loop(){}
```

Watch LED blink 5 times

“while” will check the condition (i < 5), if not zero, it will execute the codes within its curly bracket until the condition (i < 5) returns a 0 (when false, will return a 0).

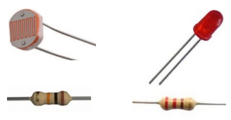
Use the Arduino IDE Software
1.Save as “do while”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

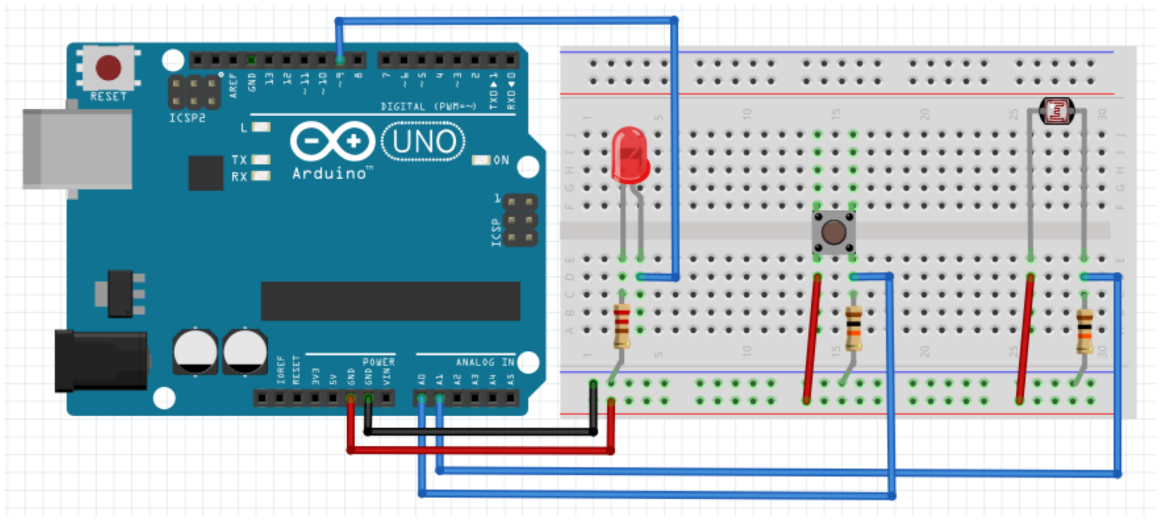
```
// Program Name = do while
#define LED_PIN 9
void setup() {
int i = 0;
  pinMode(LED_PIN, OUTPUT);
  do {
    digitalWrite(LED_PIN, HIGH); delay(500);
    digitalWrite(LED_PIN, LOW); delay(500);
    i = i+1;
  } while ( i < 5 );
}
void loop(){}
```

Watch LED blink 5 times

“do while” will repeat codes within its curly bracket until the condition (i < 5) returns a 0 (when false, will return a 0)

Program Name uses the LDR (Light Dependent Resistor).
Switch ON LED light based on the surrounding brightness. LED brightness (uses PWM), brighter LED for darker environment





Use the Arduino IDE Software
1.Save as “auto night light”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

```
// program name = auto night light
#define LED_PIN 9
#define LDR_PIN A1
#define LEVEL_0 150  // Darkest
#define LEVEL_1 300  //
#define LEVEL_2 500  // Brightest
#define LDR_READ_TIME 2000 // Read Delay Timer

int oldLevel;
unsigned long oldMillis;

void setup() {
  pinMode(LED_PIN, OUTPUT);
  Serial.begin(9600);
  Serial.print("\n\nStarting system...\n");  // 5 quick blinks
  for (int i=5;i>0;i--) {
    Serial.print("for ");Serial.println(i);
    digitalWrite(LED_PIN, HIGH); delay(500);
    digitalWrite(LED_PIN, LOW); delay(500);
  }
  oldMillis = millis();
  oldLevel = calcLevel();
  Serial.print("\nStart Level ");Serial.print(oldLevel);
  displayLed(oldLevel);
  Serial.println("");
}
```

```
void loop(){
int newLevel;
unsigned long newMillis;
  newMillis = millis();
  if ((newMillis-oldMillis) > LDR_READ_TIME ) { // Timer elapsed
    oldMillis = newMillis;
    Serial.print(LDR_READ_TIME);Serial.print(" milisecond(s) elapsed. ");
    newLevel = calcLevel();
    if ( newLevel != oldLevel ) {
      Serial.print("Level Changed from ");
      Serial.print(oldLevel);
      oldLevel = newLevel;
      Serial.print(" to ");
      Serial.print(newLevel);
      displayLed(newLevel);
      Serial.println("");
    } else {
      Serial.print("Same Level ");
      Serial.println(oldLevel);
    }
  }
}
void displayLed(int bLevel) {
  // Currently use a single PWM powered LED for light source
  // You can change the code for a different light source for each case
  switch (bLevel) {
    case 0: // darkest surroundings
      Serial.print(", LED PWM 255");
      analogWrite(LED_PIN,255);
    break;
    case 1:
      Serial.print(", LED PWM 170");
      analogWrite(LED_PIN,170);
    break;
    case 2:
      Serial.print(", LED PWM 60");
      analogWrite(LED_PIN,60);
    break;
    case 3: // brightest surroundings
      Serial.print(", LED PWN 0");
      analogWrite(LED_PIN,0);
    break;
    Default: break;
  }
}
int calcLevel() {
int pinValue;
  pinValue = analogRead(LDR_PIN);
  if (pinValue < LEVEL_0) return 0;
  if (pinValue >= LEVEL_0 && pinValue < LEVEL_1) return 1;
  if (pinValue >= LEVEL_1 && pinValue < LEVEL_2) return 2;
  if (pinValue >= LEVEL_2) return 3;
}
```

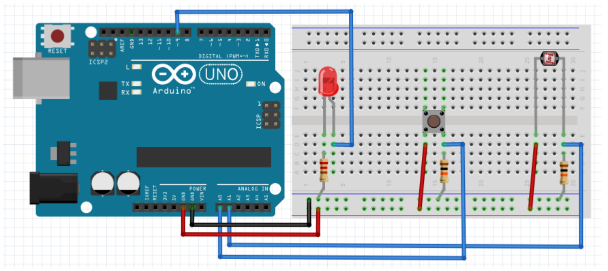
Try Cover the LDR from light and see the effect on LED, if you cant see the difference, watch the Serial Monitor Output

analogWrite(LED_PIN,255);
It sounded like “analog output”, actually it is digital output but in PWM Mode. Only those pins labeled with **~ symbol** have PWM ability. PWM Value range from 0 to 255 (whole numbers only)
If PWM value 255, during a process cycle, the LED remains ON all the time
If PWM value is half of 255, during a process cycle, the LED will go ON/OFF alternately, LED have 50% ON and 50% OFF. Causing the LED to appear dimmer
IF PWM value is quarter of 255, during a process cycle, the LED have 25% ON and 75% OFF...

The tactile switch can toggle morse code SOS flasher to stop or start. Also equiped with a LDR sensor, in the dark it will flash LED(arduino pin “9”), in the day time it will use active speaker(arduino pin “8”)

Use the Arduino IDE Software
1.Save as “start stop SOS”
2.Key in the codes below into the Arduino IDE Software Code Editor and Upload

Add Hardware. Active speaker for sound, short leg to GND, long leg to arduino pin “8”



```
// program name = start stop SOS
#define BUTTON_PIN A0
#define LDR_PIN    A1
#define SPEAKER_PIN 8
#define LED_PIN     9
#define MC_DOT    300
#define MC_DASH   900  // DASH = DOT x 3
#define MC_PART   300  // PART SPACE = DOT x 1
#define MC_LETTER 900  // LETTER SPACE = DOT x 3
#define MC_WORD   2100 // WORD SPACE = DOT x 7
unsigned int bLetter[12];    // increase this if not enough
unsigned int bMessage[100];  // increase this if not enough
unsigned int messagePos;
unsigned int ledState;
unsigned int oldButtonState;
unsigned long oldMillis;
char switchedOn;
char mcTargetPin;
void setup() {
 pinMode(LED_PIN,OUTPUT);
 pinMode(SPEAKER_PIN, OUTPUT);
 for (int i=0; i<100;i++) bMessage[i] = 0;
 mcMessageAddLetter('S'); mcMessageAddLetterSpace();
 mcMessageAddLetter('O'); mcMessageAddLetterSpace();
 mcMessageAddLetter('S'); mcMessageAddWordSpace();
 oldButtonState = 0;mcOutputReset();
}
void loop() {
unsigned long newMillis;
unsigned int newButtonState;
  newButtonState = digitalRead(BUTTON_PIN);
  if (newButtonState != oldButtonState) {
    if (newButtonState == 1) {
      delay(50); // added to filter "noise"
      switchedOn = (switchedOn=='Y')?'N':'Y';
      if (switchedOn == 'Y') {
         mcOutputReset();
      } else {
        ledState = 0; digitalWrite(mcTargetPin,ledState);
      }
    }
    oldButtonState = newButtonState;
  }
  if (switchedOn == 'Y') {
    if (bMessage[messagePos] > 0) {
      newMillis = millis();
      if ( (newMillis-oldMillis) >= bMessage[messagePos]) {
        ledState = (ledState==0) ? 1:0;
        digitalWrite(mcTargetPin,ledState);
        messagePos++;
        oldMillis = newMillis;
      }
    } else {
      ledState = 0; digitalWrite(mcTargetPin,ledState);
      mcOutputReset();
    }
  }
}
```

```
void mcOutputReset() {
 oldMillis  = millis();
 messagePos = 0;
 switchedOn = 'Y';
 if (analogRead(LDR_PIN) < 200 ) {
   mcTargetPin = LED_PIN;
 } else {
   mcTargetPin = SPEAKER_PIN;
 }
 ledState = 1; digitalWrite(mcTargetPin,ledState);
}

void mcMessageAddLetterSpace() {
unsigned int startPos;
  startPos = 0;
  while (bMessage[startPos] != 0) startPos++;
  bMessage[startPos] = MC_LETTER;
}

void mcMessageAddWordSpace() {
unsigned int startPos;
  startPos = 0;
  while (bMessage[startPos] != 0) startPos++;
  bMessage[startPos] = MC_WORD;
}

void mcMessageAddLetter(char mcLetter) {
unsigned int partPos;
unsigned int startPos;
  mcBuildLetter(mcLetter);
  startPos = 0;
  while (bMessage[startPos] != 0) startPos++;
  partPos = 0;
  while (bLetter[partPos] != 0) {
    bMessage[startPos] = bLetter[partPos];
    startPos++;
    partPos++;
  }
}

void mcBuildLetter(char mcLetter) {
  for (int i=0; i<12;i++) bLetter[i] = 0;
  switch (mcLetter) {
  case 'S':
    bLetter[0] = MC_DOT;
    bLetter[1] = MC_PART;
    bLetter[2] = MC_DOT;
    bLetter[3] = MC_PART;
    bLetter[4] = MC_DOT;
  break;
  case 'O':
    bLetter[0] = MC_DASH;
    bLetter[1] = MC_PART;
    bLetter[2] = MC_DASH;
    bLetter[3] = MC_PART;
    bLetter[4] = MC_DASH;
  break;
  default: break;
  }
}
```