SEICHE 2022
Basic Arduino Programming

Instructor: Paul Frommeyer

www.paulfrommeyer.com

Corporate Sponsor: DXC Technology

Lesson Plan Overview

Lesson 1 – Intro and Setup
[may require 2 classes]

- Introduction to class format
- Overview of lesson plan
- Presentation format (monitor, camera, screen, whiteboard)
- Review of microcontrollers and types of boards
- SEICHE LED display architecture
  - ESP8266 pinout
  - High level architecture

Lesson 2 – Laptop operation review – Windows and Linux

- Inventory of USB drives
- Installation of Arduino IDE software
- Installation of CH340/ESP8266 serial port drivers (Windows only)
- Control panel/settings location
- Home directories and folder hierarchy
- Arduino file locations
- Search functions
- (Windows) Device Manager
- (Linux) Konsole
- Copying flash drive contents [critical]
- Open questions and issues
- **IDE essentials**
- Starting the Arduino IDE
- Basic Arduino sketch (program) structure
- Loading example sketches
- Loading and configuring new boards
- Connecting boards
- Identifying the microcontroller serial port
  - Linux
  - Windows

Lesson 3 – Libraries, Sketch structure, Serial Monitor, Variables, Binary Number System Pt1

- Libraries
- Sketch structure (A note on brace formatting)
- The serial port monitor
- Printing to the serial port monitor
- Variables and the assignment operator
- Binary number system Pt. 1.

Lesson 4 – Expressions, Conditionals, Blocks and Functions

- Arithmetic Expressions and Operators
- Incrementing and Decrementing Variables
- Truth Values in C++
- The If-Then Statement
- Code Blocks
- Functions

Lesson 5 – Binary Images, Arrays, Characters, Strings, Loops

- Loading Binary Images
- Arrays
- Characters and Character Codes
- Strings
- Conditional Loops Part 1

Lesson 6 – Loops (cont.), LED Matrix Displays, Nested Loops Advanced Functions, Binary Numbers Part 1

- For-Next Loops
- SPI Peripherals
- Using a MAX7219 LED Matrix Display
- Lighting and clearing individual pixels
- Advanced Functions
- Nested Loops

Lesson 7 – The Binary Number System (may take 2 lessons)

- Numerals vs numbers
- Review: the base 10 system and digit place values
- New: the base 2 system and digit place values
- Bits and bytes and nybbles
- Binary addition and subtraction

Lesson 8 – Producing Sound

- Formatting printed output in Serial Monitor
- Shifting and exponents
- Bitwise operations and masking
- Displaying text on the LED matrix display
- Review of sound wave theory
- Analog vs Pulse Width Modulation
- Producing sound tones with an Arduino microcontroller

Lesson 9 – Reading Analog and Digital pins

- Millis
- Reading buttons
- Debouncing buttons
- Reading analog values from a potentiometer

Lesson 10 – The I2C Bus and Peripherals

- I2C Bus Operation
- Initializing the I2C bus
- Accessing an I2C temperature sensor
- Real Time Clocks
- Accessing a DS3231 RTC

Lesson 11 – NTP and Text Management

- Numeric to ASCII Conversions
- Time representations and conversions
- Network Time Protocol
- Displaying the time on an LED matrix display

Lesson 12 – Text Management

- sprintf revisited
- Text Effects
- Changing the default font
- Using multiple display zones

Lesson 12 – sprintf redux, Text Effects, Changing Fonts, Multiple Display Zones

- Class Exercise - sprintf() redux
- Class Exercise - Text Effects
- Introduction to fonts – bitmap fonts
- Including supplemental code with #include
- Classroom Exercise – Using Different Fonts
- MD_Parola Display Zones
- Classroom Exercise – Using multiple display zones

sprintf() Redux

Below is an example of how sprintf() can be used in actual code

```
void setup()
{
  Serial.begin(9600);
  sprintf(buffer,"%i @@ %c @@ %d @@ %f @@ %e @@ %u @@ %s", i, c, d, f, u, sometext);
  Serial.println(buffer);
}
```

CLASSROOM EXCERCISE

- Load Sketch 12A into your displays
- Now, modify the sprintf statement in the sketch to do the following:
  - Display a byte along with a string (text of your choice)
  - Display an integer with at least three leading zeros
  - Display a float with two decimals of precision
  - Display a float in exponential notation
  - Display all of the above together (you may need to change the character buffer size to handle all of the output)

Note: sprintf format reference can be found at the end of the presentation

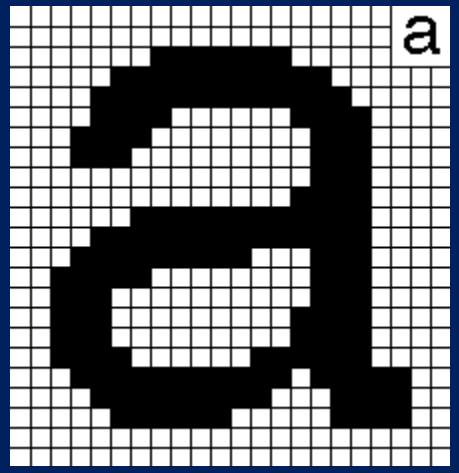
Class Exercise – Text Effects

- The MD_Parola library offers a number of text effects that can be used to put text on the display and then remove it
- The effect used when displaying text is called the entry effect
- The effect used when removing text is called the exit effect
- You can see these in your code, they are `PA_LEFT, PA_LEFT`
- Below is a list of just some of the Parola entry and exit effects; they can be used symmetrically or in any combination
- Try them out!
- The full list can be found here: https://majicdesigns.github.io/MD_Parola/_m_d___parola_8h.html#acf3b849a996dbbe48ca173d2b0b82eda

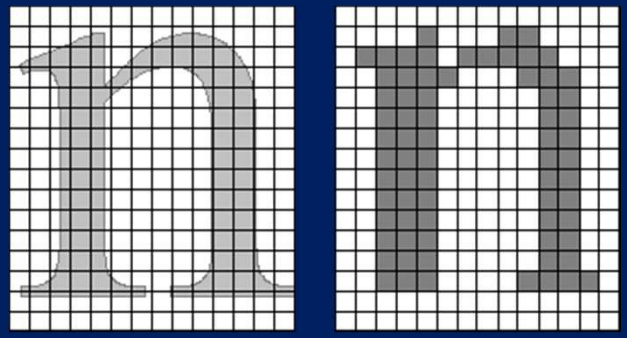
PA_NO_EFFECT	PA_FADE	PA_OPENING	PA_GROW_UP
PA_PRINT	PA_DISSOLVE	PA_OPENING_CURSOR	PA_GROW_DOWN
PA_SCROLL_UP	PA_BLINDS	PA_CLOSING	
PA_SCROLL_DOWN	PA_RANDOM	PA_CLOSING_CURSOR	
PA_SCROLL_LEFT	PA_WIPE	PA_SCROLL_UP_LEFT	
PA_SCROLL_RIGHT	PA_WIPE_CURSOR	PA_SCROLL_UP_RIGHT	
PA_SLICE	PA_SCAN_HORIZ	PA_SCROLL_DOWN_LEFT	
PA_MESH	PA_SCAN_VERT	PA_SCROLL_UP_RIGHT	

Introduction to Fonts - Bitmap

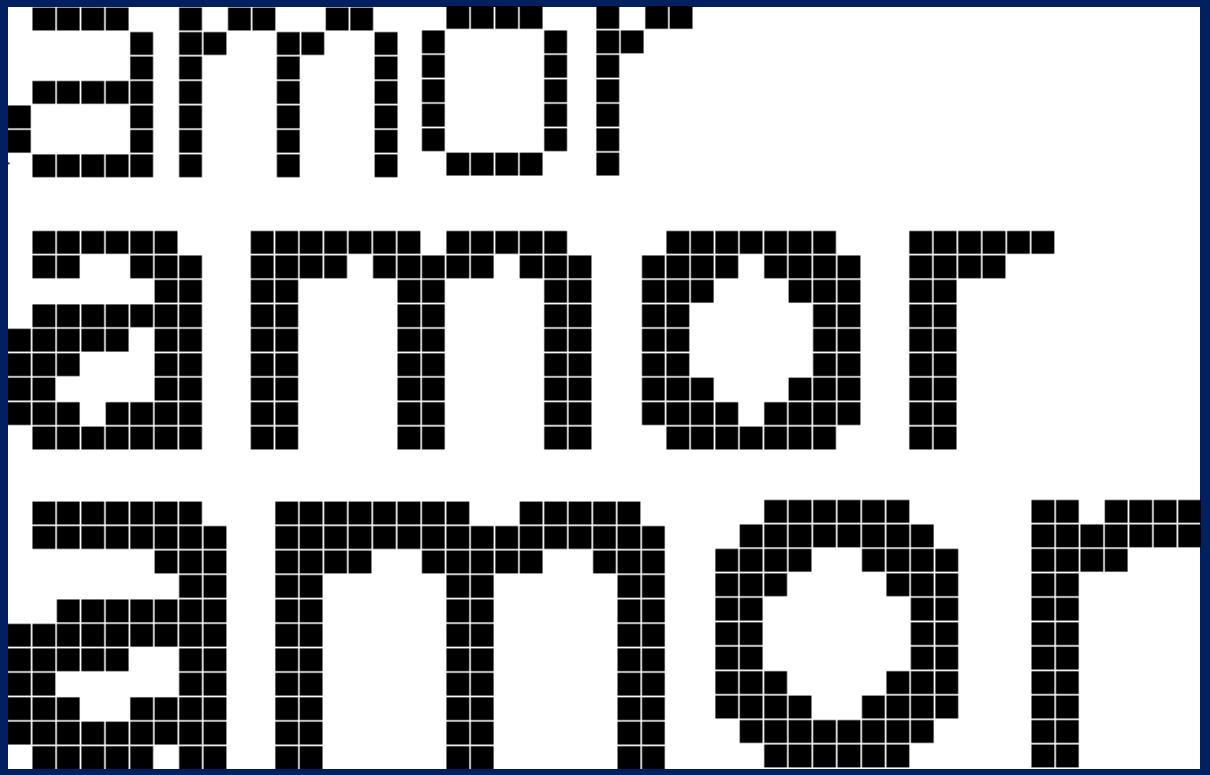
- Bitmap fonts specify which pixels in the character are on or off
- As a result, bitmap fonts are fixed in size, and not rescalable
- While outline fonts are normally sized by *points*, bitmaps are often sized by *pixels*



A 23x22 bitmap character



Comparison: outline vs 14x16 bitmap



A word in different bitmap sizes

Including supplemental code

- The #include statements we've used so far, at the beginning of our sketches, are how we have incorporated *libraries* into our sketches
- However, it's easy to incorporate *our own* code in the same way
- The IDE will search the same folder as the sketch for any files we include by using quotes "". These so-called *header files* must have a suffix of .h
- Important note: the .h is *not* used when specifying the file in the IDE!
- The following statement will incorporate the code from the file **myheaderfile.h** into your sketch:

```
#include "myheaderfile"
```

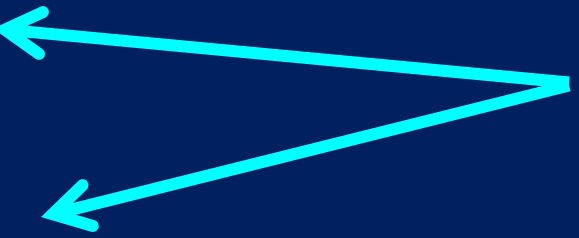
Classroom Exercise
Using Different Fonts

- Load SKETCH12B
- There are a number of font header files in your SKETCH12B directory; have a look at their filenames in your file explorer.
- Now modify your sketch to use a *different* font than the one already specified in SKETCH12B

```
#include <SPI.h>
#include <MD_MAX72xx.h>
#include <MD_Parola.h>
#include"Parola-boldFont.h"

...

 pmx.setFont(parola_boldFont);
```



Note that the font header file begins with a capital “P” and the font name itself begins with a lowercase “p”!

MD Parola Display Zones

- Normally, MD_Parola treats the entire display as a single display zone, numbered zone zero (0).
- The number of 8x8 display modules contained in zone zero is specified by the MAX_DISPLAYS definition at the beginnging of your sketches
- However, with Parola it is possible to break up a series of MAX7219 8x8 LED matrix displays into *multiple* zones.
- This allows completely different data, fonts, and event text effects to be applied to different areas of a display

Classroom Exercise
Using Multiple Display Zones

- Let's see how that works! Load SKETCH12C
- Notice the section of code where the different zones are defined
- Now modify your code as follows:
  - Apply different fonts to each zone
  - Change the entry and exit effects for each zone

```
void setup()
{
  // put your setup code here, to run once:
  pmx.begin(2);
  pmx.setZone(0,0,1);
  pmx.setZone(1,2,3);
  pmx.setZoneEffect(0, true, PA_FLIP_LR);
  pmx.setZoneEffect(0, true, PA_FLIP_UD);
  pmx.setZoneEffect(1, true, PA_FLIP_LR);
  pmx.setZoneEffect(1, true, PA_FLIP_UD);
  pmx.setFont(1,parola_boldFont);
}

void loop()
{
  if(pmx.displayAnimate())
  {
    pmx.displayZoneText(0,"SEICHE",PA_CENTER,50,0,PA_SCROLL_RIGHT,PA_SCROLL_RIGHT);
    pmx.displayZoneText(1,"2022",PA_CENTER,50,0,PA_SCROLL_LEFT,PA_SCROLL_LEFT);
  }
}
```

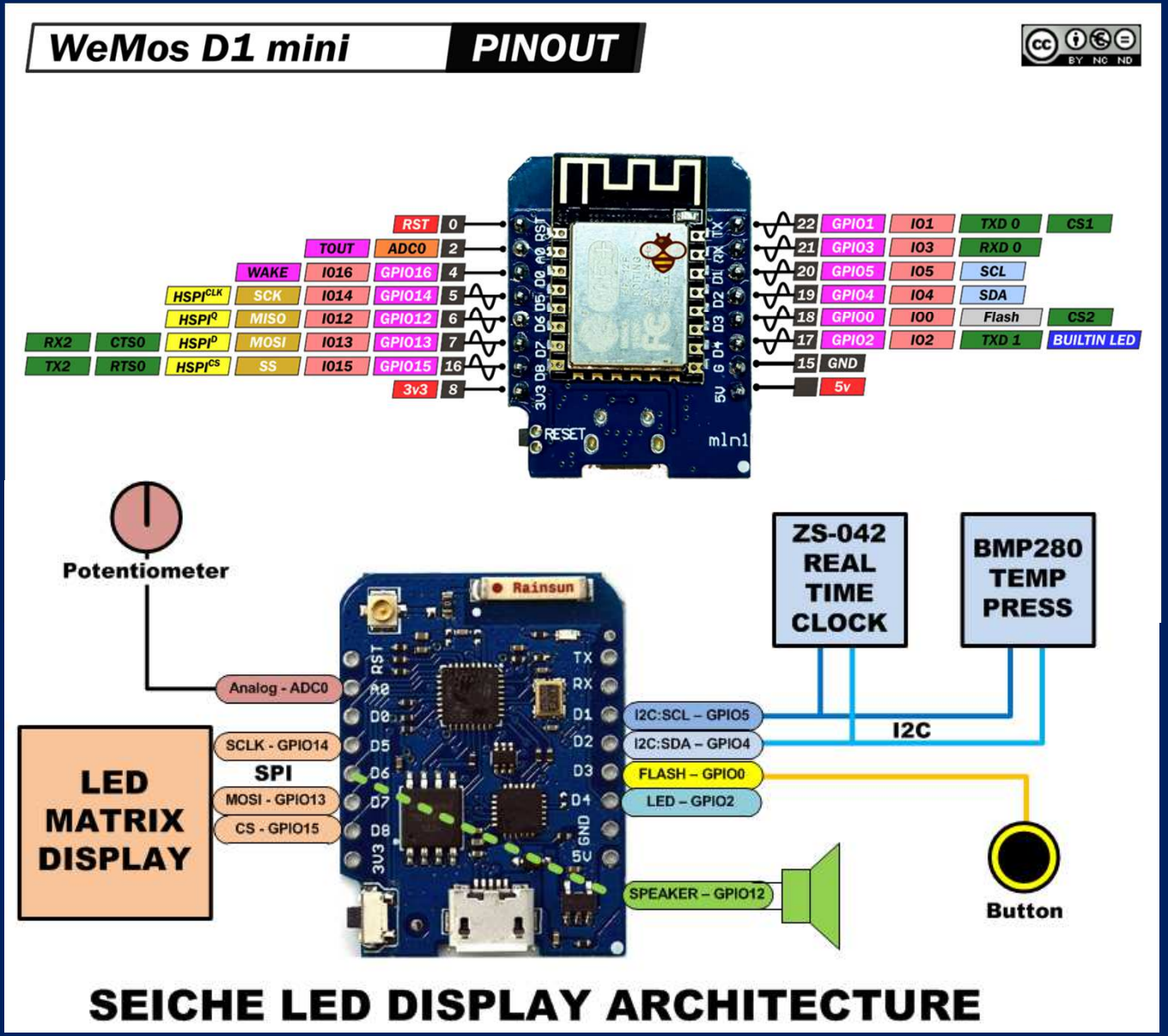
END OF AUTUMN SEMESTER
END OF BASIC ARDUINO PROGRAMMING!

- **NO HOMEWORK ☺**
- **IF YOU ARE NOT RETURNING NEXT SEMESTER, KEEP YOUR USB DRIVES and DISPLAYS!**
- **IF YOU ARE RETURNING, PLEASE RETURN YOUR USB DRIVES**
- **YOU MAY KEEP YOUR DISPLAYS OVER BREAK**

Be sure to tune in for Intermediate Programming next semester!

- Creating bitmap fonts
- Accessing information on the global Internet
- Serving web pages with ESP8266
- Network communication with ESP8266 and MQTT

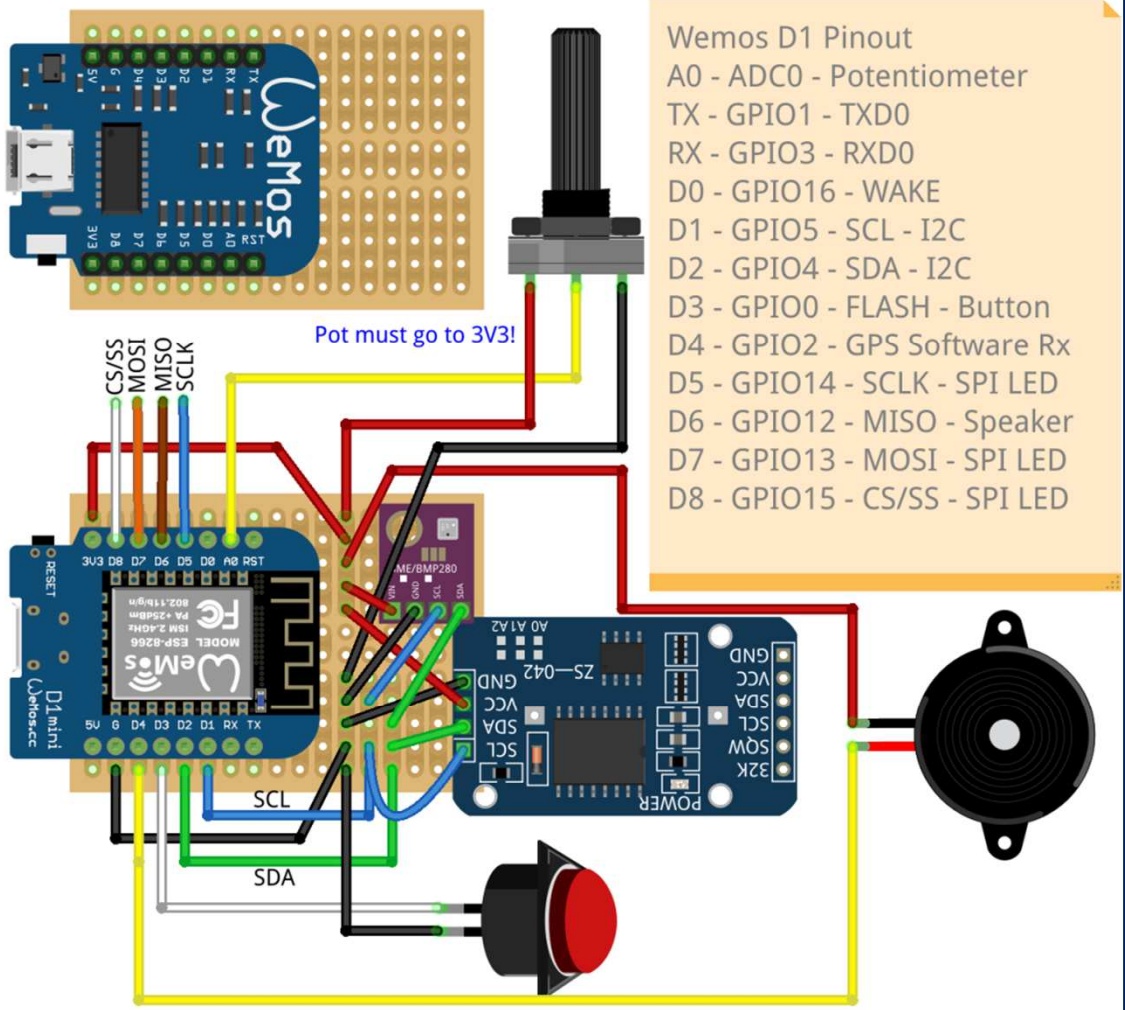
LESSON REFERENCE



LESSON REFERENCE

Pin Assignment Notes
GPIO16/D0 - HIGH at boot - No interrupt, no PWM or I2C - Unused
GPIO2/D4 - HIGH at boot - Input pulled up, output to onboard LED - - probable GPS RX software serial
GPIO12/D6 - Piezo Speaker (not used in SPI LED Matrix)
GPIO[12],13,14,15/D6,D7,D5,D8 - MISO,MOSI,SCLK,CS - SPI
GPIO4,5/D2,D1 - SDA,SCL - I2C
ADC0/A0 - Analog Input - Potentiometer 3.3V divider
GPIO0/D3 - Input pulled up - FLASH button, boot fails if pulled low - button to ground

SPI - LED matrix : 12,13,14,15
I2C - RTC,BMP280 : 4,5
Serial RX - GPS : 16 SS
Input pullup with interrupt - Button : 0
Piezo Speaker : 2
Analog Input - Potentiometer : ADC0



Wemos D1 Pinout
A0 - ADC0 - Potentiometer
TX - GPIO1 - TXD0
RX - GPIO3 - RXD0
D0 - GPIO16 - WAKE
D1 - GPIO5 - SCL - I2C
D2 - GPIO4 - SDA - I2C
D3 - GPIO0 - FLASH - Button
D4 - GPIO2 - GPS Software Rx
D5 - GPIO14 - SCLK - SPI LED
D6 - GPIO12 - MISO - Speaker
D7 - GPIO13 - MOSI - SPI LED
D8 - GPIO15 - CS/SS - SPI LED

`sprintf()` Format Strings

- `sprintf()` format strings contain conversion specifiers that specify how each individual variable is to be converted
- `sprintf()` conversion specifiers have the following general syntax (all begin with a percent-sign):

  %[flags][minimum field width][.][precision][length][conversion character]

  - % - special token that indicates the start of a conversion specifier
  - Flags – these modify the behavior of the specification
  - Minumum field width – as it says on the tin; this the minimum number of characters to be converted
  - . – The period is a separator between field width and precision
  - Precision – means one of the following depending on the variable type and conversion specifier
    - The maximum number of characters to be generated from a string
    - The number of digits after the decimal point for type float conversions (e, E, or f)
    - The zero-filled minimum number of digits for an integer
  - Conversion character – A single character which determines the output type of the conversion specifier; a single character that specifies the type of output format for the corresponding data or variable

`sprintf()` Conversion Specifiers

Below are the general conversion specifiers and what they do.

Specifier	What it does
d, i	int - integer; signed decimal notation
o	int – unsigned octal (no leading zero)
x, X	int – unsigned hexadecimal, no leading 0x
u	int – unsigned decimal
c	int – single character, after conversion to unsigned char
s	char * - characters from string are printed until \0 (NULL) or *precision* is reached
f	double – decimal notation of form [-]mmm.ddd where number of decimals is specified by precision; precision of zero (0) suppresses the decimals altogether
e, E	double - exp notation; default precision of 6, 0 suppresses
g, G	double – Use %f for <10^4 or %e for >10^4
p	void * - print output as a pointer, platform dependent
n	Number of characters generated so far; goes into output
%	No conversion, put a % percent sign in the output

`sprintf()` Conversion Specifiers

Below are the flags and what they do.

Flag	What it does
-	Left justification
+	Always print number with a sign
spc (space)	Prefix a space if first character is not a sign
0 (zero)	Zero fill left for numeric conversions
#	Alternate output form depending on conversion character o – first digit will be zero x or X – 0x or 0X (respectively) prefixed to non-zero results e, E, f, g and G – Output will always have a decimal point g and G – trailing zeroes will never be removed