EECS 2031 3.0 A
Software Tools

Week 2: September 12, 2017

Chapter 2 of K&R

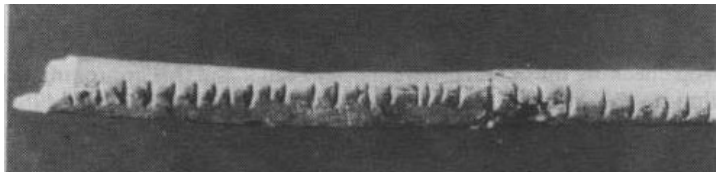
- Primitive types in C along with operators that act on them.
- The full definition of C is given in Appendix A. Look there for answers to specific questions about syntax.

High level structure of a C program

- Multiple files defining different functions, variables and constants.
- Program execution starts at main
- Traditional flow control (sequencing, selection, iteration).
- Local and global data storage, both on the stack and in the heap.
- Data storage is type defined.
    - And this is where we start.

How would you represent numerical information in a digital computer?

Perhaps the fundamental question in computer hardware design



Lebombo tally stick 35,000 years old

Historically: take some symbol, have N of them -> represent N

Place-value notation

- Value is based on a (typically small) set of symbols and the order in which they appear
- 123_{10} = 1 x 10^2 + 2 x 101 + 3 x 10^0
- 1011_2= 1 x 2^3 + 0 x 2^2+ 1 x 2^1 + 1 x 2^0

Binary Computers

- Two states (binary)
  - 1 (on, true)
  - 0 (off, false)

Larger numbers

3	2	1	0
0	0	0	0
0	0	0	1
0	0	1	0
0	0	1	1
0	1	0	0
0	1	0	1
0	1	1	0
0	1	1	1
1	0	0	0
1	0	0	1
1	0	1	0
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	0
1	1	1	1

Bit patterns

- For n bits, how many possible patterns?
- Assuming unsigned numbers (0…k), what numbers would we represent?
- Do we miss any numbers in the range?

3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10
1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

That's great, but what about signed numbers?

Sign value

Sign bit

3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	0
1	0	0	1	-1
1	0	1	0	-2
1	0	1	1	-3
1	1	0	0	-4
1	1	0	1	-5
1	1	1	0	-6
1	1	1	1	-7

-0!

Two’s complement

3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7

Positive numbers are the same

Two’s complement

3	2	1	0	

For negative numbers, invert the bits and add 1
(so for 1 = 0001 -> 1110+1 = 1111)
Note that for 8 = 1000 = 0111+1=1000
So you get from -8..+7

1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Two’s complement

3	2	1	0	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	-8
1	0	0	1	-7
1	0	1	0	-6
1	0	1	1	-5
1	1	0	0	-4
1	1	0	1	-3
1	1	1	0	-2
1	1	1	1	-1

Invert bits and add 1

Ganging bits together

- Bit (1 or 0)
- Nibble (4 bits) - 16 values (-8..+7)
- Byte (8 bits) - 256 values (-128..+127)
- Word (16 bits) - 65536 values (-32768..+32767)
- Word (32 bits) - 4294967295 values (-2147483648..+214483647)

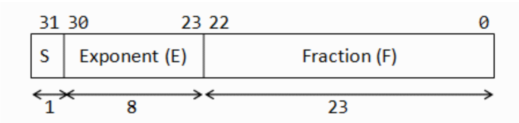
And in C

- char - signed byte
- unsigned char - unsigned byte
- short - signed 16 bits (two bytes)
- unsigned short - unsigned 16 bits (two bytes)
- int - signed 16 or 32 bits
- unsigned int - unsigned 16 or 32 bits
- long - signed 32 bits
- unsigned long - unsigned 32 bits
- Constants are signed int's unless explicitly long 0L or unsigned 0u
- Character constants (as int’s) are surrounded by single quotes. ‘a’

That’s great, but what about floating point numbers?

- So (thinking about the theory)
  - Unsigned int’s are an approximation to the natural numbers
  - Signed int’s are an approximation to integers
  - Floats (and doubles) are approximations of the Real numbers.
- How many natural numbers are there?
- How many real numbers are there?

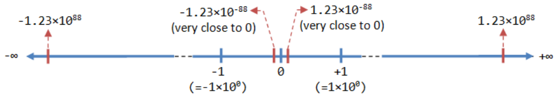
Floating point numbers



32-bit Single-Precision Floating-point Number

S - sign bit
E - exponent in 2’s complement -126..+127 (other patterns reserved)
F - fraction bit plus implicit leading 1

Exponent patterns reserved for 0, NAN, +/-INF
Fraction bit is negative powers of 2



Floating-point Numbers (Decimal)

And in C

- float - 32 bit
- double - 64 bit
- Constants are doubles unless explicitly floats 0.0f versus 0.0.

Declaring variables

Why would a compiler want you to declare variables before use?

Syntax

- Similar to Java (or perhaps better to say Java is similar to C).
- ANSI C (pedantic) requires variables to be declared at the start of a block {}, Newer ‘C’s are less restrictive.

```
int x, y, z;
float z=0.0f;
```

See A8.2 and A8.8

```
#include <stdio.h>

int main()
{
   int x = 2;
   int y = 3;
   int z = x / y;
   int t = y /x;
   printf("z is %d t is %d\n",z,t);
}
```

```
#include <stdio.h>

int main()
{
   int x = 2;
   int y = 3;
   int z = x / y;
   int t = y /x;
   printf("z is %d t is %d\n",z,t);
}
```

```
z is 0 t is 1
```

```
#include <stdio.h>

int main()
{
   float x = 2;
   float y = 3;
   float z = x / y;
   float t = y /x;
   printf("z is %f t is %f\n",z,t);
}
```

```
#include <stdio.h>

int main()
{
   float x = 2;
   float y = 3;
   float z = x / y;
   float t = y /x;
   printf("z is %f t is %f\n",z,t);
}
```

```
z is 0.666667 t is 1.500000
```

Mixed mode

- C will silently promote and silently (implicitly) convert when it has to.
  - This will surprise those Java programmers out there.

```
# include <stdio.h>

int main()
{
   float temp = 15.0;

   printf("Temperature is %f\n",(9/5) * temp + 32);
}
```

What does this output?
What implicit type conversions did C do?

```
# include <stdio.h>

int main()
{
   float temp = 15.0;

   printf("Temperature is %f\n",(9/5) * temp + 32);
}
```

What does this output?
What implicit type conversions did C do?

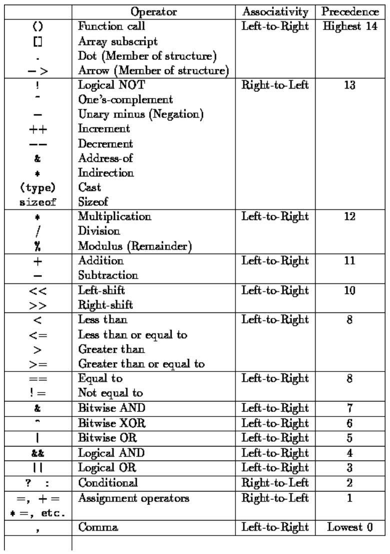
```
Temperature is 47.000000
```

Identifiers in C

- Defined in A2.3
- Lower case and upper case are different.
- First character must be an underscore _ or letter, other characters can be letters, underscores or numbers.
- At least the first 31 characters are compared when testing for equality.
- A least the first 6 characters are compared when linking across files and case may be ignored.
- There are a number of reserved words (all are lower case) and are given in A2.4

Some C operators

- Arithmetic
  - +,-,*,/,%
- Relational - return an int (0 or not zero)
  - <, >, ==, !=, >=, <=
- Logical - return an int (0 or not zero)
  - &&, ||, !
- Binary
  - &, |, ~, <<, >>, ^



	Operator	Associativity	Precedence
() [] . −>	Function call Array subscript Dot (Member of structure) Arrow (Member of structure)	Left-to-Right	Highest 14
! ~ − ++ −− & * (type) sizeof	Logical NOT One's-complement Unary minus (Negation) Increment Decrement Address-of Indirection Cast Sizeof	Right-to-Left	13
* / %	Multiplication Division Modulus (Remainder)	Left-to-Right	12
+ −	Addition Subtraction	Left-to-Right	11
<< >>	Left-shift Right-shift	Left-to-Right	10
< <= > >=	Less than Less than or equal to Greater than Greater than or equal to	Left-to-Right	8
== ! =	Equal to Not equal to	Left-to-Right	8
&	Bitwise AND	Left-to-Right	7
^	Bitwise XOR	Left-to-Right	6
|	Bitwise OR	Left-to-Right	5
&&	Logical AND	Left-to-Right	4
||	Logical OR	Left-to-Right	3
? :	Conditional	Right-to-Left	2
=, += *=, etc.	Assignment operators	Right-to-Left	1
,	Comma	Left-to-Right	Lowest 0

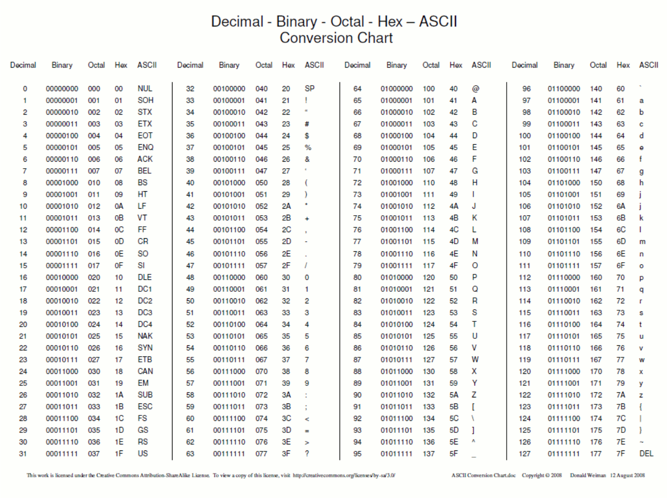
Table 5.1: Precedence and Associativity Table

What is a string?

(There is no String type in C)

In C

- A ‘string’ is an array of characters terminated by a null (0) character.
- Characters are single bytes.
  - The syntax ‘a’ defines a signed int with the value 97.
- The syntax “abcde” is a short form for an array of char’s of length 6 with the symbols ‘a’, ‘b’, ‘c’, ‘d’, ‘e’, ‘\0’ in it.



Decimal - Binary - Octal - Hex – ASCII
Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	“	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	‘	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	j
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

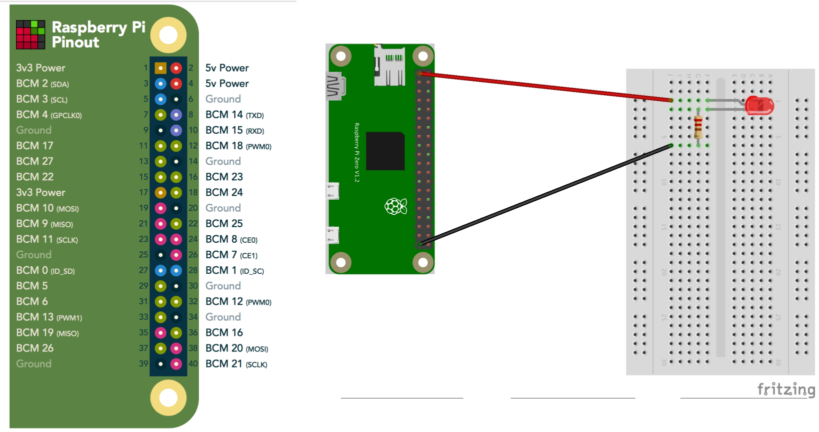
Gotcha’s

- Unlike Java, C does not specify order of evaluation completely
  - z = f() + g() : which is evaluated first, f() or g()?
- Conversions with unsigned numbers are often subtle
  - -1L > 1UL ?
- Use of -Wall is (almost) never wrong as it will alert you to some of these features.

Next week's lab

- Deals with the problem of writing very simple code to blink an LED and deal with button press input.
- The goals of the lab
  - Write some simple C code
  - Learn how to link to external libraries
  - Learn how to connect external things to a Raspberry Pi

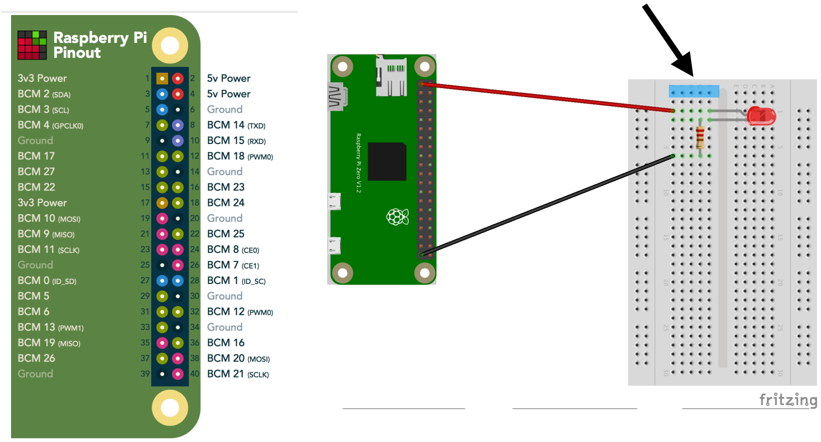
GPIO



Turning a LED on

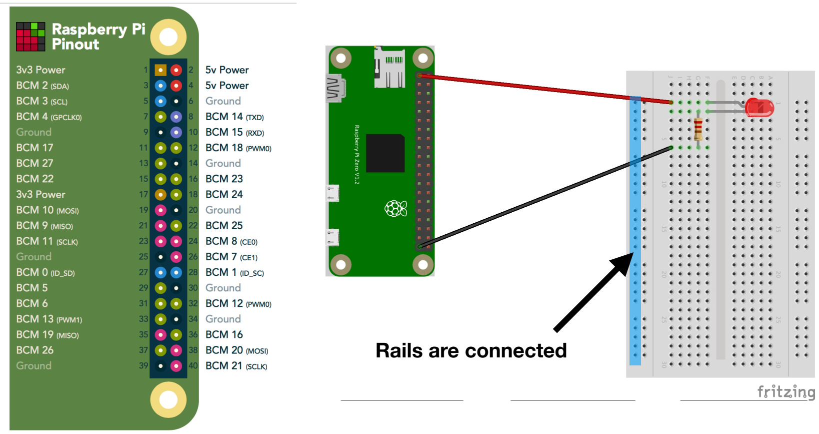
GPIO

Rows are connected



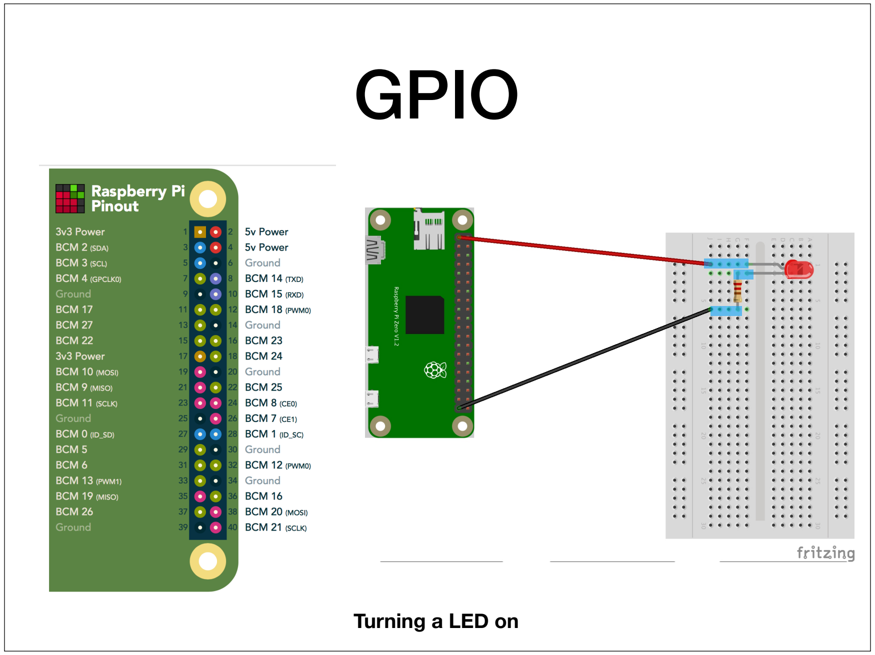
Turning a LED on

GPIO



Rails are connected

Turning a LED on

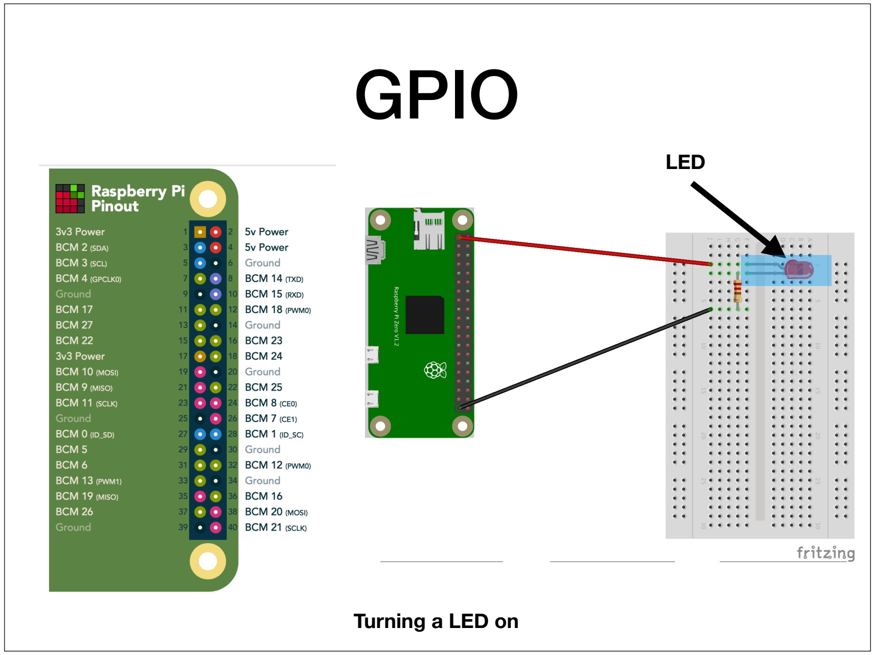


GPIO

Raspberry Pi Pinout

3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)

Turning a LED on

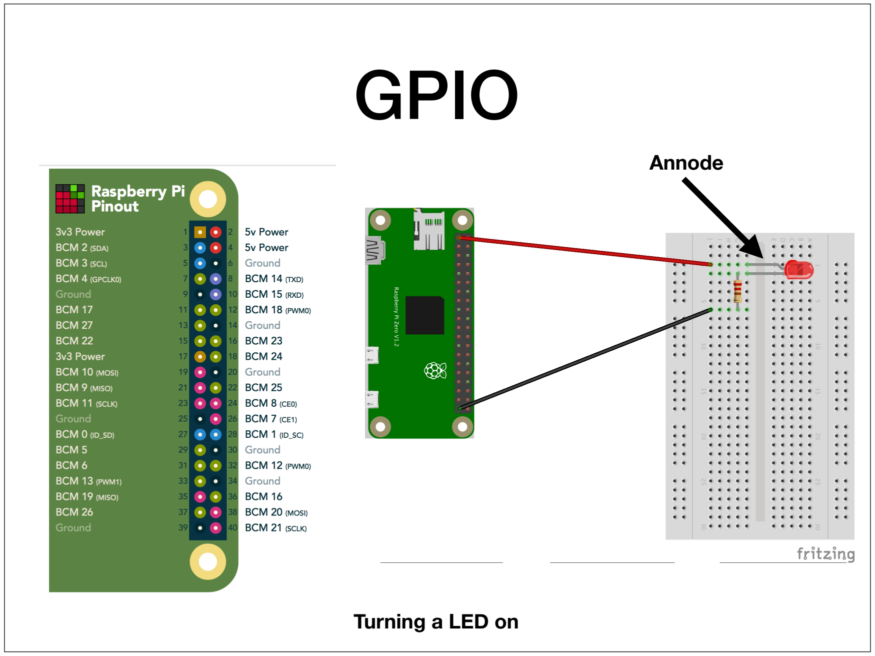


GPIO

Raspberry Pi Pinout

3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)

Turning a LED on

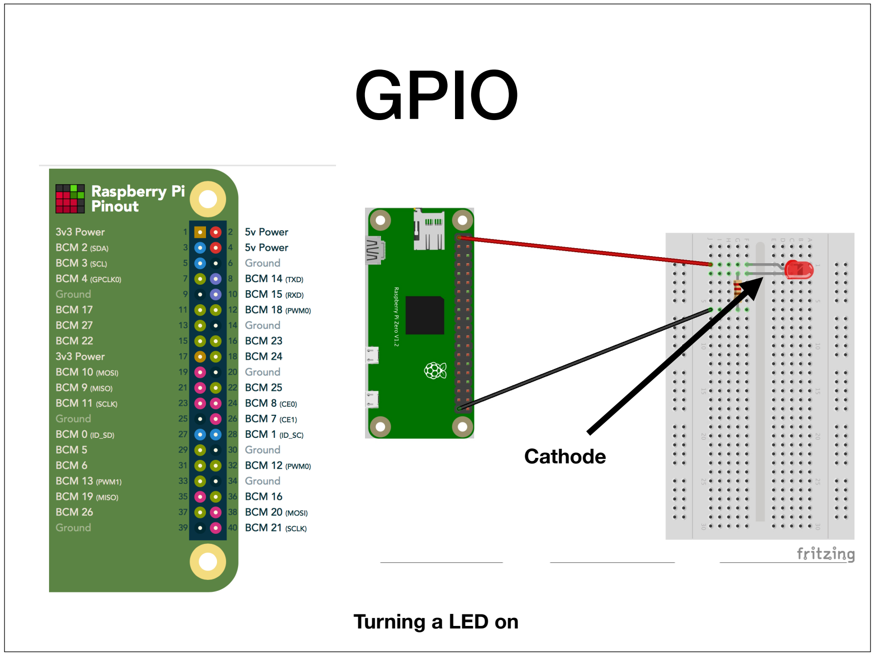


GPIO

Raspberry Pi Pinout

3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)

Turning a LED on



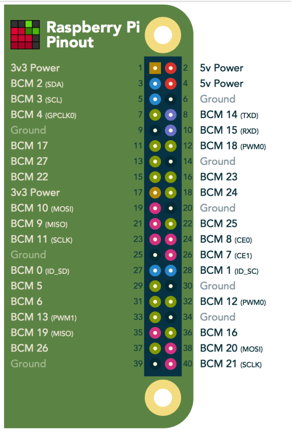
GPIO

Raspberry Pi Pinout

3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)

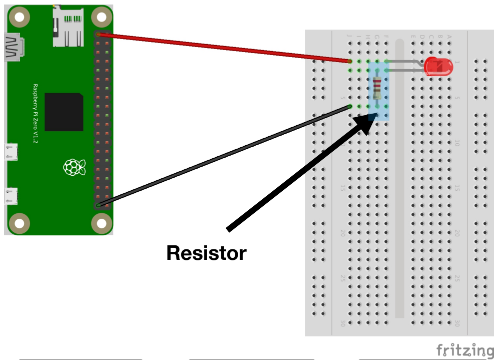
Turning a LED on

GPIO



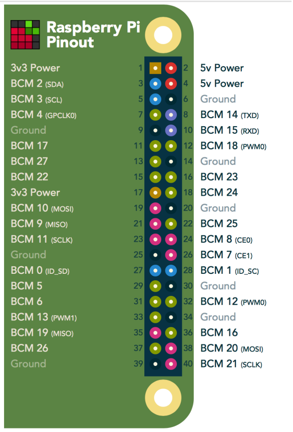
Raspberry Pi Pinout

3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)



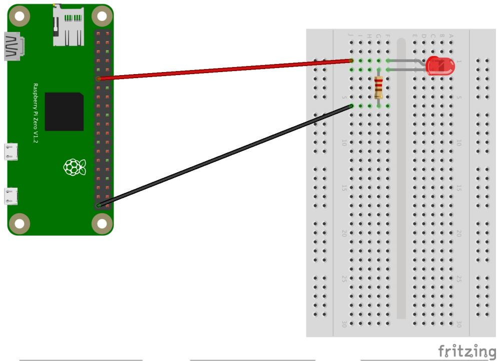
Turning a LED on

GPIO



Raspberry Pi Pinout

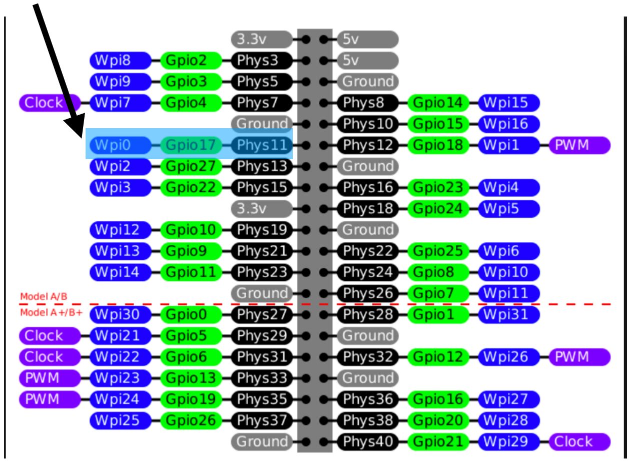
3v3 Power	1	2	5v Power
BCM 2 (SDA)	3	4	5v Power
BCM 3 (SCL)	5	6	Ground
BCM 4 (GPCLK0)	7	8	BCM 14 (TXD)
Ground	9	10	BCM 15 (RXD)
BCM 17	11	12	BCM 18 (PWM0)
BCM 27	13	14	Ground
BCM 22	15	16	BCM 23
3v3 Power	17	18	BCM 24
BCM 10 (MOSI)	19	20	Ground
BCM 9 (MISO)	21	22	BCM 25
BCM 11 (SCLK)	23	24	BCM 8 (CE0)
Ground	25	26	BCM 7 (CE1)
BCM 0 (ID_SD)	27	28	BCM 1 (ID_SC)
BCM 5	29	30	Ground
BCM 6	31	32	BCM 12 (PWM0)
BCM 13 (PWM1)	33	34	Ground
BCM 19 (MISO)	35	36	BCM 16
BCM 26	37	38	BCM 20 (MOSI)
Ground	39	40	BCM 21 (SCLK)



Turning a LED on through software

GPIO

Pins have different names from different organizations

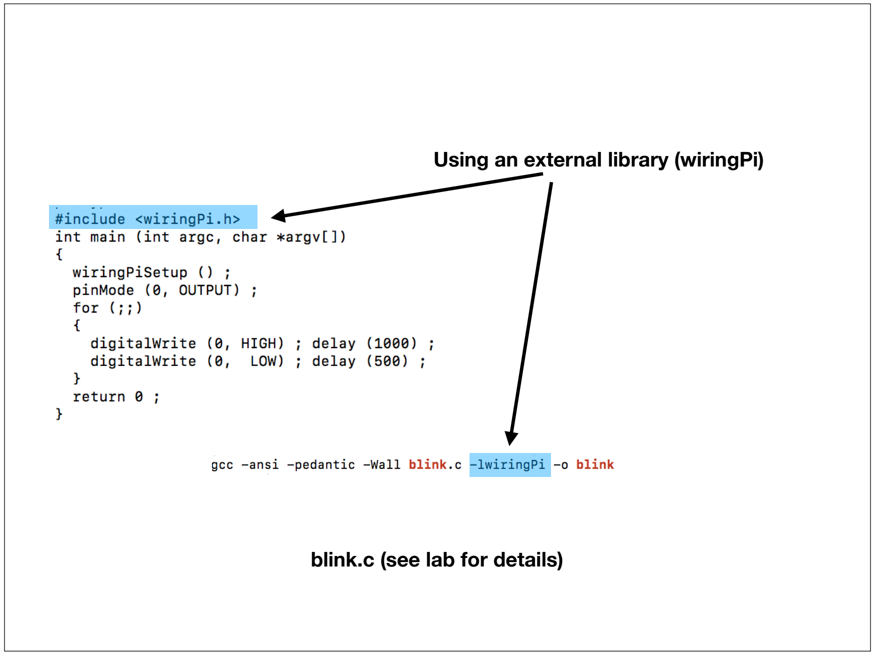


			3.3v	5v			
	Wpi8	Gpio2	Phys3	5v			
	Wpi9	Gpio3	Phys5	Ground			
Clock	Wpi7	Gpio4	Phys7	Phys8	Gpio14	Wpi15	
			Ground	Phys10	Gpio15	Wpi16	
	Wpi0	Gpio17	Phys11	Phys12	Gpio18	Wpi1	PWM
	Wpi2	Gpio27	Phys13	Ground			
	Wpi3	Gpio22	Phys15	Phys16	Gpio23	Wpi4	
			3.3v	Phys18	Gpio24	Wpi5	
	Wpi12	Gpio10	Phys19	Ground			
	Wpi13	Gpio9	Phys21	Phys22	Gpio25	Wpi6	
	Wpi14	Gpio11	Phys23	Phys24	Gpio8	Wpi10	
Model A/B			Ground	Phys26	Gpio7	Wpi11	
Model A+/B+	Wpi30	Gpio0	Phys27	Phys28	Gpio1	Wpi31	
Clock	Wpi21	Gpio5	Phys29	Ground			
Clock	Wpi22	Gpio6	Phys31	Phys32	Gpio12	Wpi26	PWM
PWM	Wpi23	Gpio13	Phys33	Ground			
PWM	Wpi24	Gpio19	Phys35	Phys36	Gpio16	Wpi27	
	Wpi25	Gpio26	Phys37	Phys38	Gpio20	Wpi28	
			Ground	Phys40	Gpio21	Wpi29	Clock

Turning a LED on

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

blink.c (see lab for details)

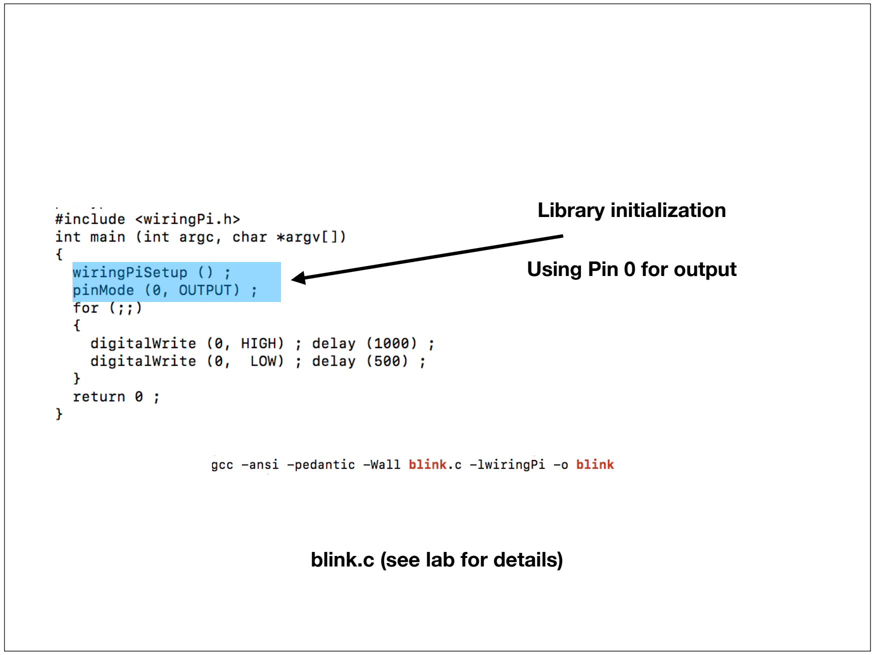


Using an external library (wiringPi)

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)



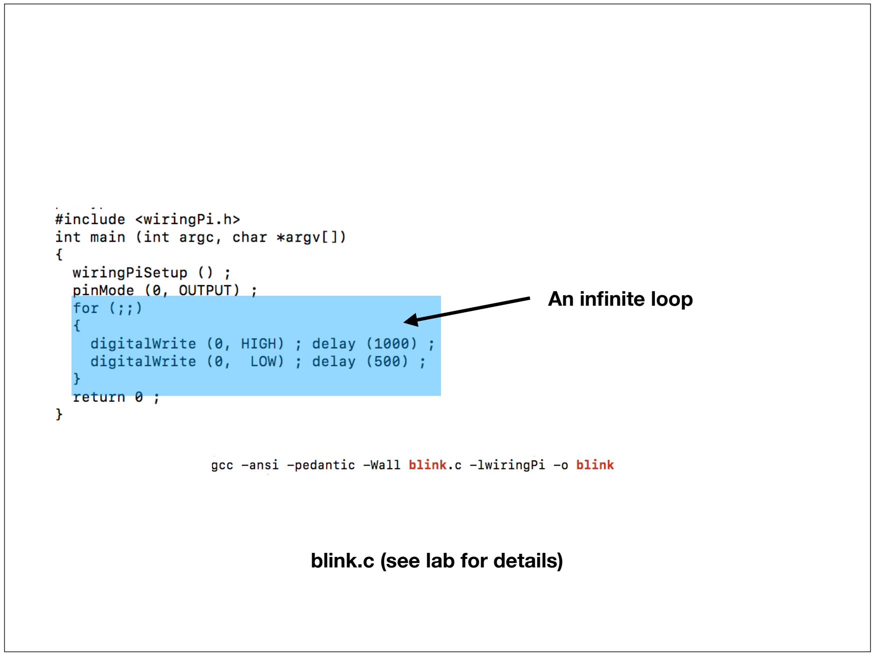
Library initialization

Using Pin 0 for output

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)

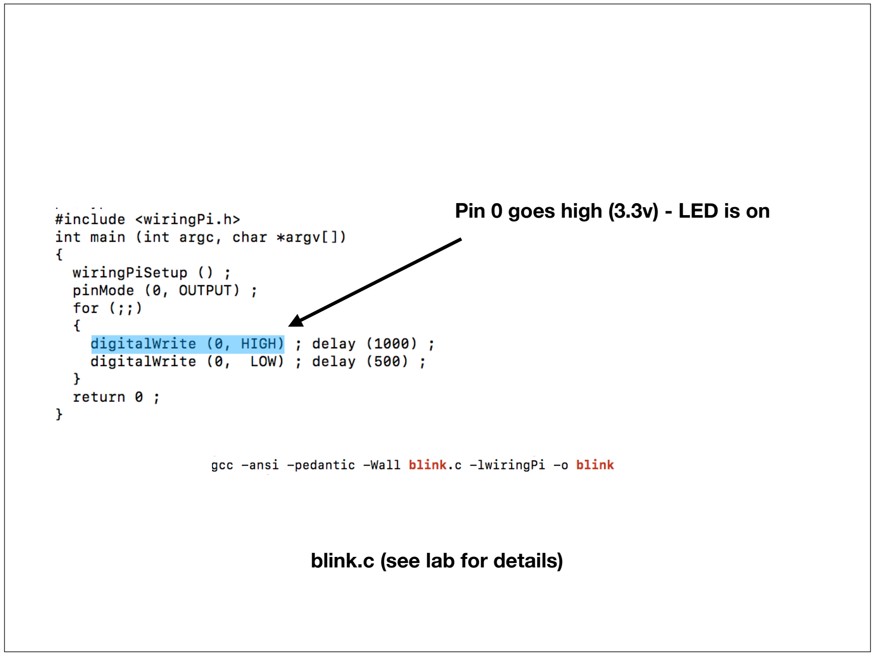


An infinite loop

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)



Pin 0 goes high (3.3v) - LED is on

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

Wait for a second (1000 msec)

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)

```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

Pin 0 goes low (0V). LED goes out

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)

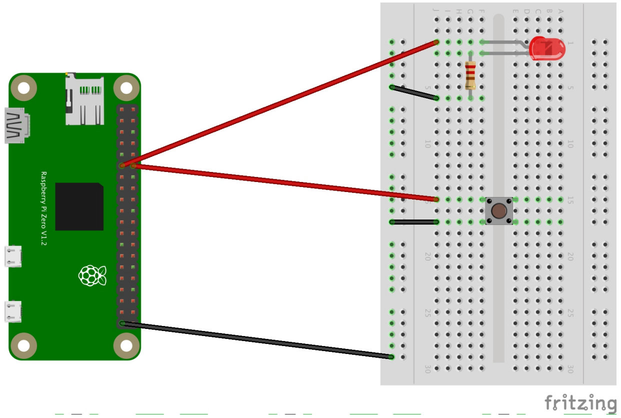
```
#include <wiringPi.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (0, OUTPUT) ;
  for (;;)
  {
    digitalWrite (0, HIGH) ; delay (1000) ;
    digitalWrite (0,  LOW) ; delay (500) ;
  }
  return 0 ;
}
```

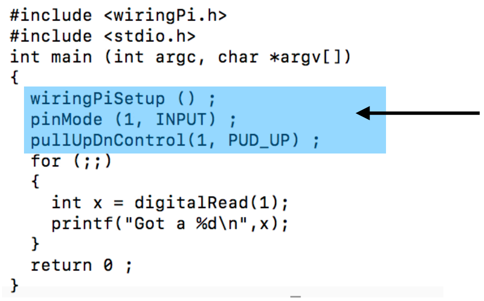
Wait for 0.5 seconds

```
gcc -ansi -pedantic -Wall blink.c -lwiringPi -o blink
```

blink.c (see lab for details)

Monitoring a button

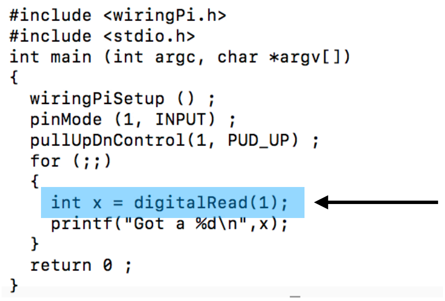




```
#include <wiringPi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (1, INPUT) ;
  pullUpDnControl(1, PUD_UP) ;
  for (;;)
  {
    int x = digitalRead(1);
    printf("Got a %d\n",x);
  }
  return 0 ;
}
```

wiringPi setup
Pin 1 as input
Pin 1 with pull up resistor

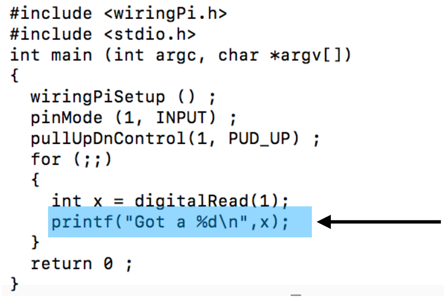
button.c



```
#include <wiringPi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (1, INPUT) ;
  pullUpDnControl(1, PUD_UP) ;
  for (;;)
  {
    int x = digitalRead(1);
    printf("Got a %d\n",x);
  }
  return 0 ;
}
```

Read pin 1

button.c



```
#include <wiringPi.h>
#include <stdio.h>
int main (int argc, char *argv[])
{
  wiringPiSetup () ;
  pinMode (1, INPUT) ;
  pullUpDnControl(1, PUD_UP) ;
  for (;;)
  {
    int x = digitalRead(1);
    printf("Got a %d\n",x);
  }
  return 0 ;
}
```

Output its value

button.c

Final thoughts

- Go to your lab.
- The code you will write in this course will continue to build on your efforts in previous weeks.
- Keep up with the reading.
- Questions?